FAKULTÄT FÜR !NFORMATIK

Mobile Agenten und Sicherheit

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Informatik

eingereicht von

Isabella Hartl
Matrikelnummer 8726132

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: O. Univ. Prof. Dipl.-Ing. Dr. techn. Dietmar Dietrich
Mitwirkung: Dipl.-Ing. Albert Treytl

Wien, 29.10.2008 _____ _____
(Unterschrift Verfasserin) (Unterschrift Betreuer)

Kurzfassung

Mobile Agenten sind Programme, die zwecks Erfüllung ihres vom Benutzer erteilten Auftrags, innerhalb eines Netzwerkes autonom von Rechner zu Rechner migrieren. Unterstützt werden sie dabei von lokalen Plattformen, ohne deren angebotene Dienste ein mobiler Agent seine Aufgabe nicht erfüllen könnte (z. B. Kommunikations- und Migrationsdienst). Sobald ein mobiler Agent seine Heimatplattform verlässt, bewegt er sich in einer potentiell böswilligen Umgebung. Seine aktuelle Wirtsplattform könnte versuchen ihn zu attackieren, indem sie seine mitgeführten Daten ausspioniert oder gar seinen Programmcode manipuliert. Gleiches gilt für die Wirtsplattform, ein mobiler Agent könnte versuchen unberechtigten Zugang zu ihren Daten oder Ressourcen zu erhalten. Aus diesem Grund ist das Thema „Sicherheit“ sowohl für mobile Agenten als auch ihre Plattformen von besonderer Wichtigkeit. Beide müssen sich gegen Angriffe absichern.

Diese Arbeit diskutiert den Begriff „Sicherheit“ und wie er in mobilen Agentensystemen umgesetzt werden kann und beschreibt, wie weit sich vorhandene Agentenstandards (FIPA und OMG) dem Thema Sicherheit und seiner Umsetzung widmen. Nach einer Analyse der möglichen Angriffe auf ein mobiles Agentensystem, werden jeweils zwei mobile Agentensysteme vorgestellt (JADE und SeMoA, Aglets und Grasshopper) und deren Sicherheitsmaßnahmen miteinander verglichen. Anschließend werden Maßnahmen zur Umsetzung der Sicherheitsanforderungen diskutiert und ihre Vor- und Nachteile besprochen. An Hand der konkreten Agentenplattform CARE für RFID wird die Umsetzung einer Kommunikationsabsicherung in einem FIPA-konformem Agentensystem vorgestellt.

Abstract

Mobile agents are autonomously acting programs. For the purpose of executing their owner orders, they migrate within a network from computer to computer. They are supported by local agencies, without their offered services a mobile agent could not accomplish its task. As soon as a mobile agent leaves its home agency, it moves within a potentially malicious environment. Its current agency could try to attack him by spying out its carried data or even by manipulating its program code. Same applies to agency a mobile agent could try to get unauthorized access to its data or resources. For this reason "security" is a topic for both of them, for mobile agents and agencies.

This work discusses the concept "security" and how it can be implemented in mobile agent systems. After introducing the security requirements of mobile agents and the cryptographic primitives needed for putting them into action, it is described how far existing agent standards (FIPA and OMG) are applying to the subject security in their implementation. After an analysis of the possible attacks on a mobile agent system, two mobile agent systems each are analyzed (JADE and SeMoA, Aglets and Grasshopper) and their security measures compared with each other. Measures for the implementation of the security requirements are then introduced and their advantages and disadvantages discussed. Finally a practical implementation in the CARE agent runtime - a FIPA compliant environment for RFID - is presented to show practical implementation of security.

Danksagung

Weihnachten Anfang der 80er Jahre entschied sich meine berufliche Zukunft. Meine Großeltern, Emma und Winfried Frank sen., kauften für ihre Firma einen der ersten „Heimcomputer“, einen Commodore PET, und präsentierten diesen der Familie. Mein Onkel, Winfried Frank jun. schrieb ein passendes Programm dazu, das jedem Familienmitglied eine besondere Eigenschaft zuordnete. Dieses Programm, das etwas über mich erzählen konnte, obwohl nur mein Vorname eingegeben wurde, weckte ein Interesse in mir, das bis heute ungebrochen ist. Den Begriff Informatik kannte ich damals noch nicht, aber von nun an Stand fest, dass mein Beruf mit Computer und Programmen zu tun haben würde. Ich möchte meinen Großeltern und meinem Onkel für dieses Schlüsselerlebnis danken. Meinen Eltern, Herta und Josef Hartl, und meinem Vater Lebrecht Angerer, danke ich dafür, dass sie dieses Ziel immer ernst genommen haben und mir mein Wunschstudium ermöglichten.

Bilal Riaz danke ich für seine Unterstützung bei den Startschwierigkeiten rund um die Implementierung in CARE. Stefan Heinl, Peter Rauscher und Barbara Staudinger danke ich für ihr Korrekturlesen; als Nichtinformatiker wissen sie nun mehr über mobile Agenten als sie wohl jemals zu wissen gehofft hatten. Albert Treytl danke ich für die Betreuung dieser Arbeit und das er auch meinen x-ten Anlauf zum Start der Diplomarbeit und zur Beendigung dieses Studiums noch immer ernst nahm. Hrn. Dietmar Dietrich danke ich für sein zeitliches Entgegenkommen rund um einen fest vorgegebenen Endtermin.

Zu guter letzt Danke an alle Freunde und Arbeitskollegen, für ihre anhaltende Unterstützung und aufmunternden Worte, die immer zur rechten Zeit kamen und für die ich immer sehr dankbar war.

Inhaltsverzeichnis

Abkürzungen

ACC	Agent Communication Channel
APSM	Agent Platform Security Manager
AMS	Agent Management System
BCD	Binary Coded Decimal
CA	Certification Authority
CBC	Cipher Block Chaining
CORBA	Common Object Broker Request Architecture
CRL	Certificate Revocation List
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
FIPS	Federal Information Processing Standards
HTTP	Hyper Text Transfer Protocol
ID	Identifikation
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IGD	Fraunhofer-Institut für Graphische Datenverarbeitung in Darmstadt
JAAS	Java Authentication and Authorization Service
JADE	Java Agent Development Framework
Java SE	Java Standard Edition
JDK	Java Development Kit
JVM	Java Virtual Machine
MAC	Message Authentication Code
MAE	Mobile Agent Encryption
MASIF	Mobile Agent System Interoperability Facility
MTS	Message Transport Service
NIST	National Institute of Standards and Technology
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
PA	Product Agent
PCC	Proof Carrying Code
PIN	Personal Identification Number
PDA	Personal Digital Assistant
PKI	Public Key Infrastructure
PMA	Plant Management Agent
PRAC	Partial Result Authentication Code
PSE	Personal Security Environment
RA	Resource Agent
REE	Reflective Execution Environment
RFID	Radio Frequency Identification
SeMoA	Secure Mobile Agents

SDK	Software Development Kit
SSL	Secure Sockets Layer
SW	Software
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTP	Trusted Third Party
VCGen	Verification Condition Generator
XML	Extensible Markup Language

1. Einleitung

Mobile Agenten sind Programme, die zwecks Erfüllung ihres vom Benutzer erteilten Auftrags, innerhalb eines Netzwerkes autonom von Rechner zu Rechner wandern. Unterstützt werden sie dabei von den lokalen Plattformen. Die Plattformen stellen Dienste zur Verfügung, die es dem Agenten erst ermöglichen seine Aufgabe zu erfüllen, etwa durch das Bereitstellen von Schnittstellen zu lokalen Ressourcen, die Unterstützung bei der Kommunikation mit anderen Plattformen oder Agenten und natürlich auch durch die Unterstützung beim Transport seines Programmcodes, da der Agent vor seiner Migration gestoppt bzw. nach seinem Eintreffen auf einer Plattform erneut gestartet werden muss. Agenten bleiben während ihrer Reise durch das Netz nicht mit ihrer Ausgangsplattform, der Heimatplattform, verbunden. Diese kann sogar inaktiv sein, etwa wenn es sich bei der Heimatplattform um ein mobiles Endgerät, wie einen *Personal Digital Assistant* (PDA), handelt.

Mobile Agenten bewegen sich, sobald sie ihre Heimatplattform verlassen haben, in einer potentiell böswilligen Umgebung. Die erfolgreiche Ausführung ihres Auftrags hängt davon ab, dass die besuchten Plattformen korrekt mit ihnen zusammenarbeiten und nicht etwa versuchen Daten auszuspähen oder gar zu manipulieren, sei es zum Zweck des eigenen Vorteils oder um anderen Agenten oder Plattformen zu schaden. Aber auch Plattformen sind gefährdet, wenn sie „fremden“ Programmen, d. h. mobilen Agenten anderer Plattformen, erlauben auf ihnen ausgeführt zu werden und dabei ihre Daten und Ressourcen zur Verfügung stellen. Böswillig manipulierte Agenten könnten versuchen der Plattform Schaden zuzufügen, indem sie z. B. vorhandene Daten manipulieren oder Ressourcen blockieren, so dass kein anderer Agent mehr damit arbeiten kann. Diese Art des Angriffs, das Blockieren angebotener Ressourcen, wird *Denial Of Service* bezeichnet.

Mobile Agentensysteme scheinen für Angreifer zu viele potentielle Angriffsziele zu bieten. Der Zwiespalt zwischen Sicherheit und mobilen Agenten zeigt sich auch in den Überschriften diverser Fachartikel. Suchte man 1999 noch nach einer Zufluchtsstätte für mobile Agenten („A Sanctuary for Mobile Agents“ [Y99]), spricht man 2000 bereits von einer Hassliebe („Software Agenten und Sicherheit – eine Hassliebe?“ [R00]); um im Jahr 2001 bereits von Satans Agenten zu reden („Programming Satan’s Agent“ [R01a]). Die vorhandenen Standardisierungsansätze rund um Sicherheit in mobilen Agentensystemen decken entweder nur einen kleinen Bereich ab oder wurden bis dato nicht weiterverfolgt.

Ziel dieser Arbeit ist es, das Thema „Sicherheit in mobilen Agentensystemen“ aus den unterschiedlichsten Blickwinkeln zu betrachten. Schwerpunkte sind die Sicherheit des Agenten bzw. die Sicherheit der Plattform. Die Sicherheit rund um den Migrationsprozess selbst ist kein Thema dieser Ar-

beit. Da auch Applets eine Form von mobilem Code darstellen und deren sicherer Transport von allgemeinem Interesse ist, findet sich zu diesem Thema bereits genügend in der allgemeinen Fachliteratur. Kapitel 2 liefert wichtige Informationen über die typischen Eigenschaften eines mobilen Agenten und beschreibt die wesentlichsten Unterschiede zwischen Agent und „normalem" Programm bzw. Agent und Client/Server-Anwendung. In Kapitel 3 werden die, für die weitere Diskussion notwendigen, kryptographischen Primitive vorgestellt und der Begriff „Sicherheit" näher diskutiert, u. a. auch aus rechtlicher Sicht. Zum Abschluss dieses Kapitels werden die vorhandenen Agentenstandards vorgestellt. Kapitel 4 analysiert die möglichen Bedrohungen, denen ein mobiles Agentensystem ausgesetzt ist. Agenten und Plattformen können sich böswillig verhalten, aber auch Angreifer von außen können versuchen das System zu attackieren. Kapitel 5 stellt vier Agentenplattformen vor und demonstriert an Hand der auf sie durchgeführten Angriffe, worauf Entwickler u. a. zu achten haben und wie Angreifer vorgehen. Kapitel 6 analysiert die Möglichkeiten zum Schutz von Agenten bzw. Plattformen und zeigt mit welchen Maßnahmen welche Sicherheitsanforderungen erfüllt werden können. Kapitel 7 stellt eine konkrete Implementierung zur Absicherung der Kommunikation zwischen zwei Agenten bzw. Plattformen vor. Das Besondere an dieser konkreten Realisierung ist, dass die verwendete Plattform CARE (*C Agent Runtime Environment*) speziell für den Einsatz auf RFID-Chips (*Radio Frequency Identification*) entwickelt wurde. Diese RFID-Chips und die darauf laufenden Agenten dienen der Produktionsüberwachung innerhalb eines Fertigungsprozesses.

2. Mobile Agenten

"All software agents are programs,
but not all programs are agents."
[FG96]

Um in den folgenden Kapiteln den Begriff Sicherheit in mobilen Agentensystemen diskutieren zu können, ist es notwendig, die Aufgaben, Ziele und Möglichkeiten eines mobilen Agenten zu verstehen. Was ist ein mobiler Agent? Wodurch unterscheidet er sich von einem herkömmlichen SW-Programm und welche Vorteile bietet ein mobiler Agent gegenüber einer Client/Server-Architektur? Und vor allem: Wieso ist das Thema Sicherheit im Zusammenhang mit mobilen Agenten von besonderer Relevanz? Der Klärung dieser Fragen und dem Vorstellen der wichtigsten Begriffe rund um mobile Agenten widmet sich dieses Kapitel.

Softwareagenten

Das Wort Agent stammt ursprünglich aus dem Lateinischen (*agere*: tun, ausführen, handeln) und bezeichnet eine Person, die im Auftrag einer anderen Person handelt und deren Interessen vertritt [DK06]. In der Softwarewelt ist ein Agent ein Computerprogramm, das autonom im Auftrag einer Person oder Organisation arbeitet. Die wesentlichsten Merkmale eines Softwareagenten nach [BR05], [NPR06] und [JW98] sind:

- **Autonomes Verhalten**
  Agenten agieren, in Abstimmung mit den vom Benutzer vorgegebenen Aufgaben, nach einem selbst erarbeiteten Plan. Die einzelnen Schritte dieses Plans müssen nicht explizit vom Benutzer angestoßen werden, ebenso wird dieser weder über jeden einzelnen Schritt informiert, noch jedesmal um Erlaubnis gebeten.
- **Soziales Verhalten**
  Agenten können untereinander oder mit ihren Benutzern kommunizieren. Diese Kommunikation kann entweder zum Zweck des reinen Informationsaustausches stattfinden, oder aber es werden dabei eigene Protokolle abgearbeitet, wie z. B. beim Aushandeln des günstigsten Preises einer bestimmten Ware.
- **Reaktionsfähigkeit**
  Agenten beobachten ihre Umgebung und können auf eintretende Ereignisse entsprechend reagieren.

- **Handlungsfähigkeit**
  Agenten reagieren nicht nur auf Ereignisse, sondern können auch selbst die Initiative ergreifen und einen eigenen Plan entwickeln. Hat ein Agent etwa den Auftrag, ein Leihauto zu bestellen und erfährt, dass die bevorzugte Automarke mit Automatikschaltung nicht zur Verfügung steht, wird er, unter Berücksichtigung des Wissens, dass sein Auftraggeber auf Automatikschaltung mehr Wert legt als auf eine bestimmte Automarke, ein Fahrzeug ähnlichen Typs, dafür aber mit Automatikschaltung, bestellen.

Mobile Agenten besitzen zusätzlich noch eine weitere, bereits in ihrem Namen hervorgehobene, Eigenschaft:

- **Mobilität**
  Mobile Agenten haben die Fähigkeit, ihr Programm gemeinsam mit Zustandsinformationen und Daten von einem Rechner (über ein Netzwerk) zu einem anderen Rechner zu transportieren [R00]. Der Vorgang selbst wird Migration genannt.

Plattformen

Damit Softwareagenten existieren und mit ihrer Umgebung und anderen Agenten interagieren können, benötigen sie gewisse Basisdienste, die über eine Laufzeitumgebung zur Verfügung gestellt werden. Diese Laufzeitumgebung wird Agentenplattform genannt. Die Plattform liefert eine homogene Zwischenschicht in einer heterogenen IT-Landschaft, so dass derselbe Agent auf verschiedenen Rechnern mit unterschiedlicher Hardware und unterschiedlichen Betriebssystemen ausgeführt werden kann. Die Ausgangsplattform, d. h. die Plattform von der aus ein mobiler Agent seine Reise durch das Netzwerk startet, wird Heimatplattform genannt. Üblicherweise können auf einem Rechner mehrere Plattformen gleichzeitig laufen, ebenso können in einer Plattform mehrere Agenten gleichzeitig ausgeführt werden.

Folgende Aufgaben werden von der Plattform unterstützt [DK06]:

- **Management**
  Mit Hilfe der Plattform können administrative Aufgaben wie das Anlegen, Starten und Stoppen von Agenten durchgeführt werden.

- **Kommunikation**
  Mit Hilfe der Plattform können Daten ausgetauscht werden, z. B. zwischen einzelnen Agenten oder zwischen einem mobilen Agenten und seiner Heimatplattform.

- **Verzeichnisse**
  Manche Plattformen bieten, ähnlich einem Telefonbuch, Verzeichnisse an, in die Agenten die von ihnen angebotenen Dienste eintragen können, so dass andere Agenten, die diese Dienste benötigen, danach suchen und darauf zugreifen können.

- **Migration**
  Die Plattform unterstützt mobile Agenten beim Wechsel von einer Plattform zur nächsten. Der Agent wird terminiert, sein Programmcode und seine Daten serialisiert und (über ein Netzwerk)

zur folgenden Plattform übertragen. Die neue Wirtsplattform empfängt Daten und Code und startet den Agenten erneut.

Mobile Agenten

Mobile Agenten sind Computerprogramme, die im Auftrag ihres Benutzers, ihres Eigentümers, in einem Netzwerk unterwegs sind, z. B. zu jenem Punkt im Netz, wo der gesuchte Dienst oder die gesuchte Information angeboten werden. Charakteristische Merkmale mobiler Agenten nach [BR05] und [R00] sind:

- Sie werden üblicherweise in heterogenen Netzwerken eingesetzt, in denen keine Aussage über die Zuverlässigkeit oder die Sicherheit der Netzwerkverbindung getroffen werden kann, wie dies z. B. bei der Verwendung mobiler Endgeräte (z. B. PDA) der Fall ist.
- Mobile Agenten können sich in einem Netzwerk selbstbestimmt von Punkt A nach Punkt B transportieren. Das Initialereignis zum Transport kommt vom mobilen Agenten selbst und nicht z. B. vom Benutzer oder vom Betriebssystem.
- Die Migration des mobilen Agenten zu einem anderen Punkt im Netz erfolgt, um auf dort vorhandene Ressourcen oder Informationen zugreifen zu können. In diesem Punkt ähneln sich mobile Agenten und Client/Server-Anwendungen sehr. Beide nutzen die auf einem entfernten Rechner vorhandenen Ressourcen. Besteht die Aufgabe darin, eine größere Menge entfernt gespeicherter Daten zu verarbeiten, so ist es effizienter und billiger, die Verarbeitungsmethode zu den Daten zu bringen (*Function Shipping*), anstatt die Daten vom Server zum Client zu transportieren (*Data Shipping*). Ist die benötigte Verarbeitungsmethode bereits vor Ort am Server vorhanden (*Stored Procedure*) mag eine Client/Server-Anwendung ausreichen. Mobile Agenten bieten aber die Möglichkeit, eigene Verarbeitungsprozeduren, die vor Ort am Server nicht zur Verfügung stehen, definieren und anwenden zu können. Beispiele dafür sind etwa die Analyse einer sehr großen meteorologischen Datenmenge nach projektspezifischen Kriterien oder das Durchsuchen von Börsedaten nach eigenen, selbst definierten, Bewertungsstrategien. [M98]
- Da mobile Agenten autonom handeln, benötigen sie auch keine ständige Netzwerkverbindung zu ihrer Heimatplattform. Handelt es sich bei der Heimatplattform um ein mobiles Endgeräten, wie z. B. PDA oder Handy, erledigt der Agent auf der Wirtsplattform seinen Auftrag und wartet dann ab, bis seine Heimatplattform wieder online ist und er auf sie zurückkehren kann. Diese Eigenschaft, dass keine ständige Verbindung bestehen muss, ist auch bei hohen Netzwerksverbindungskosten von Vorteil.

Mobile Agenten lassen sich, abhängig davon wie weit sie sich von ihrer Heimatplattform entfernen, in zwei Arten unterteilen:

- *One-hop* Agenten (auch *single-hop* Agenten genannt)
  One-hop Agenten verlassen ihre Heimatplattform nur für einen Sprung, d. h. sie migrieren von ihrer Heimatplattform aus nur zu ihrer ersten und einzigen Zielplattform.
- *Multi-hop* Agenten
  Multi-hop Agenten migrieren von ihrer ersten Zielplattform aus noch zu weiteren Wirtsplattformen.

Bei beiden Arten von mobilen Agenten kann die Reiseroute des mobilen Agenten vom Eigentümer fix vorgegeben oder vom Agenten frei wählbar sein.

Sicherheit

Ein großer Unterschied zwischen einem mobilen Agentensystem und einer Client/Server-Anwendung besteht darin, dass ein mobiler Agent die Sicherheitsdomäne seines angestammten Rechners verlässt und mit seinem Programmcode und seinen Daten in die Sicherheitsdomäne des Zielrechners eindringt, während in einer Client/Server-Anwendung sowohl Client als auch Server geschützt aus ihrer angestammten Domäne heraus arbeiten. Bei Client/Server-Anwendungen konzentrieren sich die möglichen Angriffe daher hauptsächlich auf den Kommunikationskanal. Bei mobilen Agentensystemen besteht nicht nur die Gefahr von Angriffen auf den Kommunikationskanal, sondern auch von Angriffen aus dem Inneren des Systems, d. h. von Plattform zu Agent und umgekehrt oder von Agent zu Agent. Ein Agent, der bereits länger unterwegs ist und mehrere Wirtsplattformen besucht hat, könnte z. B. auf seinem Weg manipuliert worden sein und daher nicht mehr vertrauenswürdig sein. Eine Plattform könnte andere Interessen als der Agent verfolgen und sich weigern diesen auszuführen oder gar seine Migration zur nächsten Plattform verhindern. [R00]

Mit Hilfe kryptographischer Maßnahmen lassen sich Angriffe verhindern oder zumindest rechtzeitig aufdecken. Der Begriff Sicherheit lässt sich in unterschiedliche Sicherheitsanforderungen unterteilen, wie z. B. Vertraulichkeit, darunter versteht man das Verbergen von Daten. Im folgenden Kapitel werden sowohl die kryptographischen Primitive vorgestellt, die bei der Abwehr von Angriffen zum Einsatz kommen, als auch die einzelnen Sicherheitsanforderungen diskutiert, die in Summe den Begriff „Sicherheit“ bilden.

3. State Of The Art

"The ultimate security is
your understanding of reality."
H. Stanley Judd

In diesem Kapitel werden gängige Arbeitsmethoden und Denkweisen rund um Kryptographie, den Begriff „Sicherheit" in der Informationstechnologie und mobile Agenten vorgestellt. Sie liefern die Grundlagen für das Verständnis und die Umsetzung der in mobilen Agentensystemen zum Einsatz kommenden Sicherheitsarchitekturen.

3.1 Kryptographische Grundlagen

Private Daten können mit Hilfe kryptographischer Techniken vor unbefugter Einsichtnahme geschützt werden. Ebenso gibt es Methoden, die es dem Empfänger einer Nachricht ermöglichen, zu erkennen, ob die Daten während ihres Transports verändert wurden. Für mobile Agenten, die sich, sobald sie ihre Heimatplattform verlassen haben, in einer potentiell böswilligen Umgebung bewegen, ist die Kryptographie ein Mittel, um ein gewisses Maß an Sicherheit zu erlangen. Im Folgenden werden einige der dabei verwendeten und für das Verständnis der Arbeit wichtigen kryptographischen Grundbausteine vorgestellt.

3.1.1 Verschlüsselungsverfahren

Um die Vertraulichkeit von Daten zu wahren, werden diese verschlüsselt; dabei werden Klartextdaten mit Hilfe eines Schlüssels in Geheimdaten umgewandelt. Es gibt zwei Arten von Verschlüsselungsverfahren, symmetrische und asymmetrische. Werden beide Verfahren miteinander kombiniert, spricht man von einer hybriden Verschlüsselung.

Symmetrische Verschlüsselungsverfahren

Bei einer symmetrischen Verschlüsselung herrscht, wie der Name schon sagt, eine Symmetrie zwischen Sender und Empfänger. Sowohl zum Verschlüsseln der Nachricht, als auch zum Entschlüsseln der Nachricht wird derselbe Schlüssel verwendet.

Die Sicherheit symmetrischer Verschlüsselungsverfahren beruht auf der Geheimhaltung der verwendeten Schlüssel. Wird ein Schlüssel öffentlich bekannt, ist die Verschlüsselung nicht mehr sicher. Für mobile Agenten, die zusammen mit ihrem Sourcecode und ihren Daten auf fremde Plattformen migrieren und dort ausgeführt werden, ist es aber schwierig Geheimnisse, in diesem Fall symmetrische Schlüssel, zu wahren. Schützt ein Agent seine bisherigen Ergebnisdaten, d. h. seine auf anderen Plattformen gesammelten Daten, mit einem symmetrischen Schlüssel, könnte eine böswillige Plattform versuchen diesen Schlüssel auszuspionieren. Einen besseren Ansatz für mobile Agenten bieten daher die im Folgenden vorgestellten asymmetrischen Verschlüsselungsverfahren.

Ein im elektronischem Zahlungsverkehr weit verbreitetes symmetrisches Verfahren ist Triple DES. Triple DES ist ein Nachfolger von Single DES, dem *Data Encryption Standard* [NIST99]. Single DES verwendet Schlüssel mit einer Länge von 56 Bit, tatsächlich wird der Schlüssel aber mit 64 Bit (8 Byte) dargestellt. Das achte Bit je Byte ist ein Prüfbit und fließt nicht in die Verschlüsselung mit ein. Single DES war jahrelang das vom National Institute of Standards, NIST, vorgeschriebene Verschlüsselungsverfahren für die USA. Auf Grund der rasanten Entwicklungen am Hardwaremarkt die Prozessorgeschwindigkeit betreffend, gilt Single DES mittlerweile als nicht mehr sicher. Mit Hilfe eines *Brute-Force*-Angriffs lässt sich ein Single-DES-Schlüssel in weniger als drei Tagen ermitteln. Unter einem *Brute-Force*-Angriff versteht man das Durchprobieren aller möglichen Werte, in diesem Fall das Durchprobieren aller Schlüssel. Sobald eine der durchgeführten Verschlüsselungen zum erwarteten Resultat führt, wurde der korrekte Schlüssel gefunden. Triple DES, eine einfache und sehr effektive Erweiterung von Single DES, gilt weiterhin als sicher. Ein Triple-DES-Schlüssel ist 2*56 Bit lang; die Verschlüsselung selbst besteht aus einer dreifachen Anwendung von Single DES. Zuerst werden die Klartextdaten mit den ersten 56 Bit des Schlüssels verschlüsselt, danach wird das Ergebnis mit den übrigen 56 Bit des Schlüssels entschlüsselt und zum Schluss wird dieses Ergebnis mit den ersten 56 Bit des Schlüssels nochmals verschlüsselt. 2001 wurde der *Advanced Encryption Standard*, kurz AES genannt, von NIST als neuer Standard definiert. Hinter AES verbirgt sich ein Algorithmus namens Rijndael, dem Sieger eines jahrelangen Evaluierungsprozesses auf der Suche nach einem Nachfolger für (Single) DES. Trotz der Einführung von AES ist die Verwendung von Triple DES weiterhin weit verbreitet und wird auch von NIST weiterhin als noch sicher eingeschätzt.

Abbildung 3.1 zeigt die graphische Darstellung einer Triple-DES-Verschlüsselung. In der Literatur werden Ver- bzw. Entschlüsselungen üblicherweise durch zwei an ihrer Spitze verbundene Dreiecke symbolisiert. Der Pfeil von oben symbolisiert die Eingangsdaten, der Pfeil nach unten die Ausgangsdaten, d. h. das Resultat der Ver- bzw. Entschlüsselung. Der seitliche Pfeil symbolisiert den zu verwendenden Schlüssel.

Symmetrische Verfahren arbeiten in Blöcken bestimmter Länge, d. h. ihre Eingangsdaten müssen aus ein bis mehreren Blöcken fixer Länge bestehen. Die Ausgangsdaten haben dieselbe Länge wie die Eingangsdaten und bestehen somit auch aus Blöcken. DES arbeitet mit einer Blocklänge von 8 Byte, d. h. die zu verschlüsselnden Daten müssen immer eine Bytelänge modulo 8 haben. Daten die kürzer sind, müssen entsprechend verlängert werden; dieses Verfahren heißt *Padding*. Es gibt unterschiedliche Methoden Daten zu *padden*, z. B. wird beim ISO-*Padding* ([ISO7816-4]) immer ein Byte hex 0x80 angehängt und, falls noch benötigt, die restlichen Bytes mit hex 0x00 aufgefüllt. Die

Padding-Daten werden, falls benötigt, vor der Verschlüsselung angefügt und müssen, um zu den ursprünglichen Klartextdaten zu gelangen, nach der Entschlüsselung wieder entfernt werden. Bei der Verschlüsselung von sehr kleinen Datenmengen führt das Anhängen der *Padding*-Bytes zu einem entsprechenden Overhead.

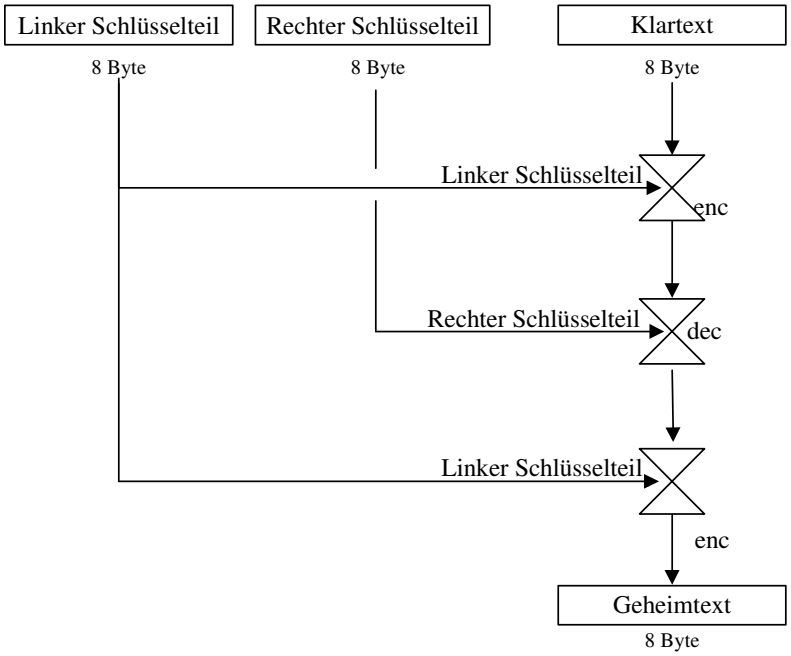


Abbildung 3.1: Triple DES

Asymmetrische Verschlüsselungsverfahren

Ein asymmetrisches Verschlüsselungsverfahren verwendet unterschiedliche Schlüssel zur Ver- und Entschlüsselung. Es gibt einen „privaten“ und einen „öffentlichen“ Schlüssel. Der private Schlüssel bleibt geheim und ist im System nur einmal vorhanden. Der öffentliche Schlüssel wird verteilt und kann im System auch mehrmals vorhanden sein. Dies ermöglicht, dass mehrere Sender unter Verwendung ein und desselben öffentlichen Schlüssels Daten verschicken können, die nur von einem einzigen Empfänger, dem Besitzer des privaten Schlüssels, entschlüsselt werden können. Umgekehrt kann der Besitzer des privaten Schlüssels Nachrichten versenden, die von allen Besitzern seines öffentlichen Schlüssels entschlüsselt werden können.

Will Bob von Alice Daten empfangen, die nur für ihn als Empfänger lesbar sein sollen, geht er wie folgt vor: Bob erzeugt ein asymmetrisches Schlüsselpaar. Das Schlüsselpaar besteht aus einem privaten Schlüssel K_{priv} und einen öffentlichen Schlüssel K_{pub}. Bob sendet den öffentlichen Schlüssel K_{pub} an Alice. Diese verschlüsselt mit diesem die, nur für Bob bestimmten, Daten. Da nur Bob in Besitz des geheimen Schlüssels K_{priv} ist, ist auch nur er in der Lage die verschlüsselten Daten K_{pub}(data) erfolgreich zu entschlüsseln. Wie aus dem Namen bereits ersichtlich ist es wichtig, dass Bob seinen geheimen Schlüssel K_{priv} vor fremden Zugriffen schützt. Nur so ist sicher gestellt, dass die Vorteile des asymmetrischen Verschlüsselungsverfahrens gewahrt bleiben. Bobs öffentlicher

Schlüssel kann beliebig verteilt werden. Ein Angreifer kann weder aus dem öffentlichen Schlüssel noch aus den verschlüsselten Daten Rückschlüsse auf den privaten Schlüssel ziehen.

Ein weit verbreitetes und in Agentensystemen häufig angewendetes asymmetrisches Verfahren ist RSA. RSA steht für die Anfangsbuchstaben der Erfinder dieses Verfahrens, Ron Rivest, Adi Shamir und Len Adleman. Die Sicherheit von RSA hängt eng mit der Schwierigkeit zusammen, große Zahlen in ihre Primfaktoren zu zerlegen. Zur Generierung des Schlüsselpaares werden zwei sehr große Primzahlen p und q benötigt. Die Multiplikation beider Primzahlen ergibt den Modulus. Der Grad der Sicherheit des Algorithmus steigt mit der Länge des Modulus, üblicherweise werden aktuell Längen von 1024 Bit (128 Byte) und höher verwendet. Aus p und q werden auch K_{priv} und K_{pub} abgeleitet. Die Details des Verfahrens sind auch für Laien relativ leicht zu verstehen und können z. B. in [B01] nachgelesen werden. Modulus und K_{priv} bilden eine Einheit, Modulus und K_{pub} ebenso. Der Modulus muss mit K_{pub} gemeinsam verteilt werden, da er für die Berechnung benötigt wird, muss aber, wie K_{pub}, nicht explizit geschützt aufbewahrt werden. [B01]

Die Länge des Modulus bestimmt auch die Länge der zu verschlüsselnden Daten bzw. die Länge des verschlüsselten Ergebnisses. Ist der Modulus 1024 Bit lang, müssen die Inputdaten ebenfalls auf diese Länge gepaddet werden bzw. in Abschnitte dieser Länge geteilt werden. Das verschlüsselte Ergebnis ist ebenfalls wieder 1024 Bit lang. Dies ist ein Nachteil von RSA. Kleine Datenmengen benötigen so in verschlüsselter Form einen relativ großen Speicherraum. Für mobile Agenten, die auf jeder Plattform Daten sammeln und diese in verschlüsselter Form mit sich nehmen, kann dies ein Problem werden. Die Größe des Agenten wächst mit jedem Migrationsschritt an. Im Vergleich zu einer symmetrischen Verschlüsselung benötigen asymmetrische Verschlüsselungen mehr Zeit für ihre Berechnungen. Dies kann bei der Verschlüsselung großer Datenmengen ein Problem darstellen.

Hybride Verschlüsselungsverfahren

Hybride Verschlüsselungsverfahren vereinen die Vorteile von symmetrischen und asymmetrischen Verfahren, d. h. Geschwindigkeit und hohe Sicherheit. Je Kommunikationssession wird ein eigens generierter symmetrischer Schlüssel S zur schnellen und sicheren Verschlüsselung der Daten verwendet, enc<S>(data). Der öffentliche Schlüssel K_{pub} des asymmetrischen Verfahrens dient zum sicheren Transport des symmetrischen Einmalschlüssels, $K_{pub}(S)$. Auf diese Weise wird garantiert, dass nur der gewünschte Empfänger, der Besitzer von K_{priv}, den symmetrischen Schlüssel entschlüsseln kann. Auf der Senderseite wird der symmetrische Schlüssel S nach der Übertragung wieder gelöscht, da dieser seine Aufgabe erfüllt hat. Vor dem nächsten Datenaustausch wird ein neuer symmetrischer Schlüssel S generiert, der asymmetrische Schüssel K_{pub} bleibt weiterhin in Verwendung.

Ein bekannter Anwender dieses Verfahrens ist das Netzwerkprotokoll SSL (*Secure Sockets Layer*).

Einmalschlüssel

Schlüssel, die nur für eine einzige Kommunikationssession verwendet werden, bezeichnet man auch als Einmalschlüssel. Das eben vorgestellte hybride Verfahren verwendet einen symmetrischen Einmalschlüssel S.

Asymmetrische Schlüssel gelten als sehr sicher und können über einen längeren Zeitraum unverändert in Verwendung bleiben. Symmetrische Verfahren sollten in Kombination mit Einmalschlüssel verwendet werden. Das Verwenden von Einmalschlüssel erhöht die Sicherheit, da es für einen Angreifer schwieriger wird, aus den abgefangenen Daten auf den Originalschlüssel zu schließen, da sich dieser je Session ändert.

Symmetrische Schlüssel werden daher oft in Kombination mit Ableitungsverfahren eingesetzt, die aus einem Generalschlüssel G einen Einmalschlüssel S ableiten. Eine einfache Möglichkeit einen Einmalschlüssel S zu erhalten, wäre z. B. eine Zufallszahl R mit dem Generalschlüssel G zu verschlüsseln. Die Zufallszahl R wird an die verschlüsselten Daten enc<S>(data) angehängt; der Empfänger, der in Besitz von G ist, berechnet sich mit Hilfe von R den verwendeten Einmalschlüssel S und entschlüsselt die empfangenen Daten.

3.1.2 Prüfsummen

Zur Absicherung von Daten werden diese häufig mit Prüfsummen versehen. An Hand der Prüfsumme über die Daten kann der Empfänger feststellen, ob die empfangenen Daten während ihres Transports manipuliert wurden. Es gibt Prüfsummenverfahren, die mit und ohne Schlüssel arbeiten.

Hashfunktionen

Eine Hashfunktion benötigt keinen Schlüssel. Hashfunktionen zählen zu den sogenannten Einwegfunkionen, d. h. es gilt als praktisch unmöglich, von ihren Ausgangswerten auf die Eingangswerte zu schließen [B01]. Kennt man nur den Hashwert, kann man nicht einmal die Länge der Eingangsdaten bestimmen, geschweige denn deren Inhalt. Des Weiteren sind gute Hashfunktionen kollisionsfrei, d. h. es ist äußerst schwer, zwei Paar Eingangsdaten zu finden, die beide denselben Hashwert als Ergebnis liefern. Sie können auf beliebig lange Datenmengen angewandt werden und liefern ein Ergebnis fixer Länge, das ein Abbild der Eingangsdaten darstellt. Ein bekannter Hashalgorithmus ist SHA-1, der Daten beliebiger Länge auf Daten der fixen Länge 20 Byte abbildet. Verändern sich die Eingangsdaten, verändern sich auch die Ausgangsdaten. [S96]

Der Sender berechnet über seine Nachricht einen Hashwert und verschickt diesen gemeinsam mit den Daten. Der Empfänger berechnet über die empfangenen Daten ebenfalls einen Hash und vergleicht das Ergebnis mit dem Hashwert des Senders. Stimmen die beiden Werte überein, kann der Empfänger davon ausgehen, dass die Nachricht unversehrt bei ihm eingetroffen ist.

Message Authentication Code

Ein *Message Authentication Code*, kurz MAC genannt, ist eine schlüsselabhängige Hashfunktion [S96].

Symmetrische Verschlüsselungsverfahren arbeiten in Blöcken bestimmter Länge, d. h. die Eingangsdaten müssen mindestens dieser Blocklänge entsprechen oder ein Vielfaches davon sein. DES z. B. arbeitet in Blöcken von 8 Byte. Verknüpft man das Ergebnis der Verschlüsselung des Blocks i mit den Eingangsdaten des Blocks i+1, bevor dieser verschlüsselt wird, durch ein bitweises Exklu-

siv-Oder, schafft man so eine Verbindung zwischen allen Blöcken. Das Ergebnis der Verschlüsselung des Blocks i+1 ist vom Ergebnis der Verschlüsselung des Blocks i abhängig. Das Ergebnis des zuletzt verschlüsselten Blocks ist somit von allen Ergebnissen der vorherigen Verschlüsselungen abhängig. Dieses Verfahren nennt man auch *Cipher Block Chaining* (CBC) (siehe Abbildung 3.2). Der erste Block wird dabei mit einem Initialisierungsvektor, der aber auch aus lauter Nullen bestehen kann, verknüpft. Der letzte Block, oder Teile davon, kann als Message Authentication Code verwendet werden, da er ein Abbild aller Eingangsdaten darstellt. Eine Veränderung der Eingangsdaten führt auch zu einer Veränderung des letzten Blocks und somit zu einem veränderten MAC Wert.

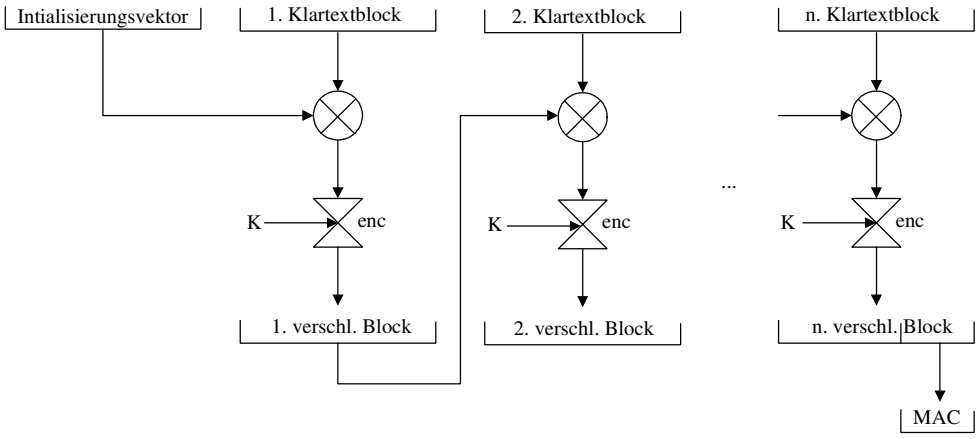


Abbildung 3.2: CBC DES *Message Authentication Code*

Transportiert ein mobiler Agent Daten und soll sichergestellt sein, dass diese nicht verändert wurden, kann er eine Prüfsumme zu diesen Daten mit sich führen oder bereits zuvor an seine Heimatpattform gesandt haben. Die Heimplattform rechnet die Prüfsumme nach und vergleicht sie mit dem vom Agenten gelieferten Wert, stimmen diese überein, kann davon ausgegangen werden, dass die Daten nicht manipuliert wurden. Der Unterschied zwischen Hash und MAC besteht darin, dass jeder, d. h. auch ein unbefugter Dritter, einen Hashwert neu berechnen kann. Ein MAC allerdings setzt voraus, dass Sender und Empfänger in Besitz des dazugehörigen symmetrischen Schlüssels sind, d. h. auch ein Angreifer würde nach einer Datenmanipulation zum Neuberechnen eines MACs diesen Schlüssel benötigen, anderenfalls fällt die Manipulation bei der Verifikation des MACs durch den Empfänger auf. Hashwerte werden daher üblicherweise zusätzlich geschützt, etwa dadurch, dass sie asymmetrisch verschlüsselt werden. Asymmetrisch verschlüsselte Hashwerte werden auch als digitale Signaturen bezeichnet.

3.1.3 Digitale Signaturen

Digitale Signaturen entsprechen einer Unterschrift. Empfänger von digital signierten Daten können an Hand der Signatur verifizieren, wer der Absender der Daten ist. Digitale Signaturen bauen auf asymmetrischen Verschlüsselungsverfahren auf. Will Bob Daten signieren, verschlüsselt er diese mit

seinem privaten Schlüssel K_{priv}. Jeder, der in Besitz von Bobs öffentlichem Schlüssel K_{pub} ist, kann nun verifizieren, dass die Daten auch wirklich von Bob stammen.

Über große Datenmengen wird vor dem Signieren häufig ein Hash gebildet, und nur der kurze Hashwert wird anschließend signiert. Der Absender A versendet die Klartextdaten gemeinsam mit der Signatur. Der Empfänger entschlüsselt die Signatur und berechnet über die mitgelieferten Klartextdaten seinen eigenen Hashwert. Wenn beide Werte übereinstimmen, d. h. der selbst berechnete Hashwert entspricht dem Inhalt der entschlüsselten Signatur, weiß der Empfänger sowohl, dass die Daten nicht manipuliert wurden, als auch, dass A der Absender der Daten war.

Digitale Signaturen kommen bei mobilen Agenten gerne zum Einsatz. Bevor ein mobiler Agent seine Reise durch das Netz antritt, kann dieser z. B. von seinem Eigentümer signiert werden. Plattformen können diese Signatur verifizieren und dem Agenten auf Grund der Identität seines Eigentümers entsprechende Rechte gewähren oder aber auch verweigern.

An dieser Stelle sei auch das Darstellungsproblem für Signaturen erwähnt. Angenommen ein Eigentümer möchte seinen mobilen Agenten signieren und sein privater Schlüssel ist z. B. auf einer Chipkarte gespeichert. Der Eigentümer des Agenten schickt nun den Hashwert des Programmcodes zur Verschlüsselung an die Chipkarte. Ein Eindringling könnte diesen Hashwert aber bevor dieser bei der Chipkarte ankommt z. B. mit dem Hashwert eines böswilligen Agenten vertauschen. In diesem Fall hätte der Eigentümer einen ihn völlig fremden Agenten signiert. Das Darstellungsproblem unterstreicht die Wichtigkeit einer guten Absicherung der Heimatplattform und der von ihr verwalteten privaten Schlüssel, diese müssen vor dem Zugriff Unbefugter geschützt werden.

3.1.4 Public Key Infrastruktur

Der Vorteil eines asymmetrischen Verschlüsselungsverfahrens besteht u. a. darin, dass die Schlüsselverwaltung einfacher ist, da nur der private Schlüssel geheim gehalten werden muss. Trotzdem entsteht bei der Verwendung dieses Verfahrens einiger organisatorischer Aufwand. Der Empfänger und zukünftige Anwender des öffentlichen Schlüssels muss überprüfen können, ob dieser öffentliche Schlüssel wirklich zur angegebenen Identität, dem Besitzer des privaten Schlüssels, gehört. Daher werden öffentliche Schlüssel, wie in [B01] beschrieben, in einer Public-Key-Infrastruktur verwaltet und verteilt.

Der private Schlüssel wird in einer persönlichen Sicherheitsumgebung (*PSE - Personal Security Environment*) abgelegt. In dieser Umgebung kann der Eigentümer des Schlüssels darauf zugreifen, Signaturen erstellen und Texte ver- bzw. entschlüsseln. Je sensibler die mit diesem Schlüssel zu schützenden Daten sind, umso sicherer sollte die PSE ausgelegt sein (Zugriffsschutz, Verwendung spezieller Kryptohardware usw.).

Zertifizierungsstellen

Vor der Verwendung eines öffentlichen Schlüssels muss sichergestellt werden zu wem dieser Schlüssel gehört. Ein böswilliger Eindringling könnte seinen eigenen öffentlichen Schlüssel unter dem Namen einer vertrauenswürdigen Komponente K in das System eingeschleust haben und wäre

somit in der Lage die an K gerichteten Nachrichten zu entschlüsseln bzw. unter Ks Namen Nachrichten zu verschicken.

Will Alice den öffentlichen Schlüssel von Bob verwenden, muss Alice zuvor verifizieren, dass dieser auch wirklich zu Bob gehört. Anderenfalls könnte ein böswilliger Eindringling Eve den öffentlichen Schlüssel von Bob durch ihren eigenen öffentlichen Schlüssel ersetzten. Eve könnte dadurch Nachrichten mit ihrem privaten Schlüssel signieren und Alice dahingehend täuschen, dass Alice glaubt, diese Nachrichten seien von Bob unterschrieben. Außerdem wäre Eve dadurch in der Lage, die von Alice an Bob ausgehenden Nachrichten zu entschlüsseln, da diese in Wahrheit mit Eves und nicht mit Bobs öffentlichem Schlüssel verschlüsselt wären.

Damit ein solcher Täuschungsversuch auffällt, ist jeder Teilnehmer eines Public-Key-Systems einer Zertifizierungsstelle, einer CA (*Certification Authority*), zugeordnet. Die CA bürgt mit ihrer eigenen Signatur für die Korrektheit der öffentlichen Schlüssel. Dies läuft wie folgt ab:

Will Bob neues Mitglied eines Public-Key-Systems werden, wendet er sich zwecks Registrierung an eine CA. Die CA überprüft Bobs Daten und weist Bob einen eindeutigen Benutzernamen zu, unter dem Bob in Zukunft auch seine Signaturen erzeugen wird. Für gewöhnlich wird auch das Schlüsselpaar bei der CA erzeugt, der private Schlüssel muss im Anschluss daran gesichert an Bobs PSE übertragen werden. Für jede Aufgabe (signieren, verschlüsseln, authentifizieren, usw.) muss ein eigenes Schlüsselpaar erzeugt werden. Der Grund dafür kann an einem Beispiel verdeutlicht werden.

Alice verwendet für Authentifikation und Signatur dasselbe Schlüsselpaar. Der böswillige Eindringling Eve gibt vor, die Authentizität von Alice prüfen zu wollen (*Challenge Response Authentication*) und schickt an Stelle der von Alice erwarteten Zufallszahl den Hashwert über ein Dokument. Alice authentifiziert die vermeintliche Zufallszahl und hat dabei aber in Wahrheit das, ihr völlig unbekannte, Dokument von Eve signiert!

Zertifikat

Zur Verifizierung der Verbindung zwischen Bob und seinem öffentlichem Schlüssel erstellt die CA ein Zertifikat. Dieses Zertifikat ist eine Aneinanderreihung von Informationen, die von der CA signiert werden. Zu diesen Informationen zählen u. a.:

1. Bobs Benutzername oder sein Pseudonym
2. Der Name der CA
3. Die Nummer der Zertifikats
4. Beginn und Ende der Gültigkeit des Zertifikats
5. Der zugeordnete öffentliche Schlüssel
6. Die Bezeichnung der Algorithmen mit denen der Schlüssel verwendet werden kann

Jedes Mitglied der PKI benötigt zur Verifizierung der *Public-Key*-Zertifikate nur den öffentlichen Schlüssel der CA.

Will Alice nun den öffentlichen Schlüssel von Bob verwenden, so fordert sie Bobs Zertifikat von der CA an. Benötigt sie Bobs Schlüssel recht häufig, kann sie ihn in ihrer PSE speichern, muss jedoch darauf achten, regelmäßig die Gültigkeit von Bobs Zertifikat zu überprüfen. Zertifikate können auch vor dem in ihnen angegebenen Ablaufdatum ungültig werden, etwa dadurch, dass der dazugehörige private Schlüssel auf Grund eines Hardwarecrashs verloren ging oder gestohlen wurde. In diesem Fall muss sich der Betroffene Zertifikatsinhaber ein neues Zertifikat mit neuem Schlüssel ausstellen lassen und das alte Zertifikat wird in eine *Certificate Revocation List* (CRL) eingetragen. Einträge in der CRL werden ebenfalls von der CA signiert. Alice sollte daher regelmäßig in der CRL nachschauen, ob das Zertifikat von Bob noch Gültigkeit hat.

Sind die mobilen Agenten bzw. deren Eigentümer und die Wirtsplattformen Mitglieder einer PKI erhöht dies die Sicherheit des mobilen Agentensystems. Dadurch kann z. B. verhindert werden, dass der öffentliche Schlüssel eines böswilligen Angreifers unter der ID eines vertrauenswürdigen Agenteneigentümers angewendet wird. Ein derartiger Manipulationsversuch würde der Plattform auffallen, da sie die zum Zertifikat gehörende Signatur der CA in diesem Fall nicht verifizieren könnte. Ebenso wird verhindert, dass vollkommen Fremde Zugriff zum System erhalten. Sobald ein Zertifikat von einer CA signiert wurde, ist die Identität der dahinterstehenden Person nicht mehr geheim und kann, im Falle eines Angriffs, über die CA ermittelt werden.

3.1.5 Administration

Der Benutzer eines Sicherheitssystems sieht dieses meistens nur in Betrieb und ahnt dabei nichts von dem dahinter liegenden Verwaltungsoverhead. Da die gängigen Verschlüsselungsalgorithmen durchwegs öffentlich bekannt sind, liegt der Erfolg eines Sicherheitssystems nicht so sehr in der Absicherung seines Sourcecodes vor neugierigen Blicken, sondern viel mehr in der sicheren Verwahrung der in den Algorithmen verwendeten Schlüssel. Auf Grund der Wichtigkeit dieser Schlüssel muss aber auch sichergestellt sein, dass, sollte einer dieser Schlüssel doch einmal korrumpiert (‚gehackt') werden, der dadurch für das System entstandene Schaden so gering wie möglich ist. Ein gut aufgebautes Schlüsselkonzept hilft somit Kosten zu sparen, dazu gehört u. a. die Verwendung von Einmalschlüssel oder die Umsetzung des Grundsatzes, dass ein Schlüssel niemals für mehrere Aufgaben verwendet werden sollte. Dient ein Schlüssel der Verschlüsselung von Daten sollte er auf keinen Fall auch zur Generierung eines MACs verwendet werden (siehe Abschnitt 3.1.1 und 3.1.2).

Des Weiteren unterliegt ein Sicherheitssystem auch einem Lebenszyklus, da bereits die Art der Erstinbetriebnahme entscheidend für die Gesamtsicherheit sein kann. Gleiches gilt für die Art der Verwaltung und letztendlich auch für den Betrieb und die Außerbetriebnahme. Bei mobilen Agentensystemen ist besonderes Augenmerk auf die Verwaltung der Plattform zu legen, da z. B. nicht sorgfältig administrierte Zugriffsrechte möglichen böswilligen Agenten Tür und Tor öffnen können.

3.2 Sicherheitsanforderungen

Um ein System als sicher einstufen zu können, muss es gewisse Grundanforderungen erfüllen. Diese Grundanforderungen werden im Folgenden vorgestellt.

3.2.1 Authentizität

„Unter Authentizität ist einerseits die eindeutige Identität einer Person und andererseits die eindeutige Zuordnung von Daten zu einer Person zu verstehen.“ [GLPR07; Seite 23] Es ist die Forderung, dass der jeweilige Gesprächspartner sich zu Beginn des Gespräches identifiziert und so dem Gegenüber die Möglichkeit gibt zu entscheiden, ob ihm vertraut werden kann oder nicht. Authentizität ist eine wichtige Sicherheitsanforderung, die auch Grundlage vieler Sicherheitslösungen ist.

Im realen Leben begegnen wir der Anforderung an Authentizität beim Leisten einer Unterschrift oder beim Vorweisen eines Ausweises. In der elektronischen Welt finden sich Äquivalente zu Unterschrift und Ausweis in der Form von Zertifikaten und elektronischen Signaturen. Elektronische Authentifizierungsprozesse basieren immer auf Geheimnissen, z. B. kann dieses Geheimnis ein privater Schlüssel K_{priv} sein, der zum Signieren eines Dokumentes dient und nur dem Dokumentenbesitzer B und sonst niemandem bekannt ist. B muss aber auch darauf achten, dass sein Geheimnis, d. h. sein privater Schlüssel, wirklich geheim bleibt. Nur unter dieser Voraussetzung kann jeder, der im Besitz des öffentlichen Schlüssels ist, verifizieren, dass besagtes Dokument wirklich von B stammt. Der öffentliche Schlüssel K_{pub} muss, wie der Name schon sagt, nicht geheim gehalten werden.

Für mobile Agenten ist es besonders schwierig Geheimnisse zu bewahren, denn trägt der Agent das Geheimnis mit sich, so wandert er damit auf seinem Weg durch das Netz von einer Wirtsplattform zur nächsten. Sollte eine dieser Plattformen böswillig sein, könnte sie versuchen das Geheimnis auszuspionieren. Maßnahmen, um dies zu verhindern, werden im Abschnitt 6.3 vorgestellt.

Migriert ein mobiler Agent von einer Wirtsplattform zur nächsten, so sollte sich sowohl die Plattform gegenüber dem Agenten, als auch der Agent gegenüber der Plattform authentifizieren. Erst wenn die Plattform dem Agenten vertraut, sollte sie diesem Ressourcen und Informationen zur Verfügung stellen, und erst wenn der Agent der Plattform vertraut, sollte er der Plattform Zugang zu seinen sensiblen Daten gewähren. [BR05]

3.2.2 Vertraulichkeit

„Unter Vertraulichkeit ist die Geheimhaltung sensibler Daten vor dem Zugriff Dritter zu verstehen“ [GLPR07; Seite 22].

Bei jeder Geldbehebung am Bankomat hat man im täglichen Leben mit der Anforderung Vertraulichkeit zu tun, denn gerade im elektronischen Zahlungsverkehr ist das Verschlüsseln sensibler Daten ein wesentlicher Punkt. So darf etwa der PIN niemals unverschlüsselt transportiert werden.

Ein mobiler Agent, der die Aufgabe hat, ein bestimmtes Produkt zu möglichst guten Bedingungen zu erwerben, sollte seine zu Grunde liegende Verhandlungstaktik gegenüber der Wirtsplattform geheim halten. Ebenso sollten die Angebote der von ihm bereits besuchten Plattformen nicht einsehbar sein, da die aktuelle Wirtsplattform entsprechend darauf reagieren könnte und ihr Angebot dadurch nur minimal günstiger als das aktuell günstigste Angebot stellen könnte. [BR05]

Wann Daten als sicherheitskritisch einzustufen sind und daher verschlüsselt gehören, hängt von der jeweiligen Anwendung ab. Als Hilfe kann folgender Gedanke dienen; würde man die Daten im realen Leben in einem verschlossenen Umschlag verwahren, so ist die Notwendigkeit zur Vertraulichkeit erfüllt. [GLPR07]

3.2.3 Integrität

In der digitalen Welt versteht man unter Prüfung der Integrität ein Verfahren, das die Echtheit von Daten Gewähr leisten und eventuelle Modifikationen durch Dritte aufdecken soll [GLPR07]. Der Empfänger einer Nachricht will erkennen, ob an ihr etwas verändert, hinzugefügt oder ausgetauscht wurde.

Früher war es üblich, Briefe mit einem Wachssiegel zu verschließen. Auf Grund des im Siegel eingeprägten Zeichens konnte der Empfänger den Absender identifizieren und auf Grund der Unversehrtheit des Siegels konnte er sich auf die Integrität des Briefinhalts verlassen. Im täglichen Leben begegnet man dem Wunsch nach Integrität z. B. darin, dass die Seiten eines unterschriftsreifen Vertrages nummeriert und zusammengeheftet werden. Mit Hilfe von Integrität kann eine Manipulation nicht verhindert werden, aber sie stellt sicher, dass eine Manipulation auffällt.

Integrität kann mit Hilfe von Hashfunktionen oder Message Authentication Codes sichergestellt werden (siehe Abschnitt 3.1.2). In der digitalen Welt wird die Forderung nach Integrität oft mit der Forderung nach Authentizität verbunden, da beides gemeinsam mit Hilfe digitaler Signaturen (siehe Abschnitt 3.1.3) realisiert werden kann. Mit Hilfe einer digitalen Signatur kann man, ähnlich einem Briefsiegel, sowohl den Absender verifizieren (siehe Abschnitt 3.2.1), als auch die Integrität der Daten sicherstellen.

Die Notwendigkeit von Integrität bei mobilen Agenten zeigt sich z. B. darin, dass der Eigentümer eines mobilen Agenten darauf vertrauen möchte, dass der Agent auch wirklich die vom Eigentümer vorgeschriebene Reiseroute einhält und diese nicht unterwegs durch eine böswillige Wirtsplattform manipuliert wird. An dieser Stelle sollte erwähnt werden, dass die vorgestellten Verfahren zwar eine Manipulation erkennen, aber nicht verhindern können! Ein weiteres Beispiel für die Notwendigkeit von Integrität ist ein mobiler Agent, der im Namen seines Eigentümers einen Kaufvertrag, z. B. über ein Flugticket, abgeschlossen hat. Sowohl Wirtsplattform als auch Agent haben Interesse daran, dass der vereinbarte Preis nicht nachträglich manipuliert wird.

„Neben den reinen Daten, die geschützt werden müssen, ist der Integritätsschutz des Agentencodes überaus wichtig.“ ([GLPR07], Seite 23). Migriert ein Agent von einer Wirtsplattform zur nächsten, muss verhindert werden, dass es dabei zu einer Manipulation seines Ausführungsverhaltens kommt. [GLPR07]

3.2.4 Verantwortlichkeit

Im realen Leben kann die Existenz eines mit Handschlag besiegelten Vertrages leicht von einem der beteiligten Vertragspartner abgestritten werden, sofern es für diesen „Handschlag“ nicht noch weitere Zeugen gibt. Daher ist es üblich, Verträge schriftlich abzuwickeln. Zur Bestätigung des Vertrages

setzt jeder Vertragspartner seine Unterschrift darunter und als Beweis für die Existenz des Vertrages erhält jeder der Beteiligten eine Kopie.

Auch im digitalen Leben der mobilen Agenten soll jede Komponente Verantwortung übernehmen und hinter den von ihr gesetzten Aktionen stehen. Kommt es zwischen Agent und Wirtsplattform nach einer erfolgreichen Verhandlung zu einem Kaufvertrag, z. B. zum Kauf des bereits zuvor erwähnten Flugtickets, darf es nicht geschehen, dass einer der beiden plötzlich behauptet, dass es diesen Vertragsabschluss niemals gab. Hier gibt es wieder die Möglichkeit der elektronischen Unterschrift, d. h. der digitalen Signatur. Wenn beide Komponenten den Vertag digital signieren, kann keiner der beiden diesen Vertragsabschluss später bestreiten. Für eine digitale Signatur muss der mobile Agent aber in Besitz eines privaten Schlüssels, d. h. eines Geheimnisses, sein. Wie bereits erwähnt ist es aber für einen mobilen Agenten schwierig Geheimnisse zu beschützen, weil er immer damit rechnen muss, auf seinem Weg durch das Netz auch auf böswillige Agenten oder Agentenplattformen zu stoßen, die ihm sein Geheimnis entreißen wollen (siehe Abschnitt 6.3). [BR05]

Aber nicht nur im „Großen“, bei Vertragsabschlüssen, muss Verantwortung übernommen werden, sondern auch im „Kleinen“. Was, wenn ein Agent schlecht programmiert wurde und auf seiner Wirtsplattform einen Schaden verursacht, etwa die Platte voll schreibt und dadurch das System zum Stillstand kommt oder eine Ressource beansprucht und nicht mehr freigibt? Die Wirtsplattform muss eine Möglichkeit haben festzustellen, wer für diese Aktionen verantwortlich ist. Kein Agent wird mit seiner Unterschrift bestätigen, dass er das System geschädigt hat, aber die Plattform sollte protokollieren, wer welche Aktionen gesetzt hat. Auch für diese Protokolldaten gelten dieselben Sicherheitsanforderungen: authentisch, vertraulich und integer. [GLPR07]

3.2.5 Verfügbarkeit

Verfügbarkeit garantiert einen zuverlässigen und prompten Zugang zu Daten und Ressourcen. Besitzt ein mobiler Agent das Recht, gewisse Aktionen durchzuführen, so soll ihm die Wirtsplattform auch darin unterstützen und ihm den Zugang zu den benötigten Daten oder Ressourcen ermöglichen. Eine böswillige Wirtsplattform könnte sich z. B. weigern, einen Agenten auszuführen oder seinen Wunsch, auf die nächste Plattform zu migrieren, ignorieren. Diese Art der Attacke nennt man *Denial Of Service*.

Verfügbarkeit kann durch Redundanz erreicht werden, z. B. durch Spiegeln der Daten auf eine zweite Festplatte, so dass ein Festplattencrash zu keinem Datenverlust und auch zu keinem Produktionsausfall führt, da die zweite Festplatte den Platz der ersten übernehmen kann. Die Verfügbarkeit kann aber auch indirekt durch die Vergabe von restriktiven Zugriffsrechten erhöht werden. Etwa dadurch, dass „wichtigeren“ Agenten rund um die Uhr und anderen „weniger wichtigen“ nur nachts Zugriff auf bestimmte Ressourcen gewährt wird. [BR05, GLPR07]

3.2.6 Anonymität

Anonymität zählt nicht zu den klassischen Sicherheitsanforderungen, widerspricht sie doch der Forderung nach Authentizität, die Grundbaustein für die meisten Sicherheitstechniken ist. Trotzdem

sind - durchaus legale - Anwendungen mobiler Agenten denkbar, bei denen der Agent den Namen seiner Heimatplattform oder die ID seines Besitzers nicht preisgeben möchte. Etwa in der Wirtschaft, wo ein zu deutlich gezeigtes Interesse an bestimmten Produkten die Konkurrenz auf sich aufmerksam machen würde. Auch können anonyme Verhandlungen für gewöhnlich einfacher und schneller vor sich gehen.

Für Mikrozahlungen, d. h. das Zahlen kleiner Beträge, wurden im elektronischen Zahlungsverkehr anonyme Chipkarten entwickelt. Mit diesen Karten lassen sich Kleinstbeträge, wie z. B. etwa Parkgebühren, an den entsprechenden Terminals schnell bezahlen, da keine PIN Eingabe notwendig ist. Ähnliches steht auch für Zahlungen im Internet zur Verfügung; Eine *Trusted Third Party* bietet virtuelle Karten an. Der Kunde erwirbt diese zu einem bestimmten Preis, der dem Wert der maximal möglichen Mikroeinkäufe entspricht. Bei jedem Kauf, etwa dem eines Musikstücks, teilt der Kunde dem Händler die Nummer seiner Karte mit und der Betrag wird von der Karte abgezogen. Der Händler tritt dabei nicht direkt mit dem anonymen Kunden in Kontakt, sondern nur mit der *Trusted Third Party*. Das langwierige Eingeben von Name und Kreditkartennummer entfällt. Durch Verwendung einer derartigen virtuellen Karte erhält ein mobiler Agent die Möglichkeit, auch anonyme Käufe zu tätigen. Da diese Karten nicht überzogen werden können, d. h. nur soviel Geld ausgegeben werden kann, wie zuvor auf die Karte geladen wurde, ist der Eigentümer des Agenten im Falle einer Attacke vor gröberen Schäden geschützt. [BR05]

3.2.7 Zugriffskontrolle

Im realen Leben verwenden wir Schlüssel, um unser Eigentum zu schützen. Wohnungen werden versperrt, um sie vor fremden Zugriff zu schützen. Niemand wäre aber erfreut, wenn der Haustorschlüssel auch alle Wohnungen des Mehrparteienhauses aufsperren könnte; umgekehrt will man es aber doch, der Schlüssel zur Wohnung soll auch das Haustor sperren können.

Für mobile Agenten und ihren Besuch auf den Wirtsplattformen gelten ähnliche Regeln. Will ein Agent seine Entscheidung für den Kauf oder Nichtkauf einer Ware von seinen zuvor getätigten Einkäufen auf dieser Wirtsplattform abhängig machen, so gewährt ihm die Plattform Zugang zu seinem Kundenkonto. Wenig angebracht wäre es, wenn er dabei gleichzeitig die Geschäftsdaten seiner Konkurrenten einsehen könnte. Daher sollten Agenten, aber auch Plattformen, mit entsprechenden Rechten und den dazugehörigen Zugriffskontrollen ausgestattet sein.

3.2.8 Protokollierung

Besonders in Systemen, die auf Sicherheit bedacht sind, ist es wichtig genau nachvollziehen zu können, wann wer welche Aktionen gesetzt hat. Das Abspeichern dieser Daten selbst muss wiederum den oben erwähnten Sicherheitsanforderungen entsprechen. Die geloggten IDs der Akteure müssen authentifiziert sein und die Logdaten müssen vor Manipulationen geschützt werden.

Protokolliert werden z. B. administrativer Aktionen, wie das Anlegen eines neuen Benutzerkontos, aber auch sicherheitskritische Datenbank- oder Ressourcenzugriffe. Wird auf einer Wirtsplattform der öffentliche Teil des Authentifizierungsschlüssels eines neuen Kunden A eingespielt, ist es von

Vorteil zu wissen, wer diese Aktion durchgeführt hat. Eventuell stellt sich später heraus, dass dieser Schlüssel in Wahrheit nicht zu A, sondern zu Spion S gehört!

3.3 Angriffe auf die Sicherheit

Angriffe auf die Sicherheit lassen sich in aktive und passive Angriffe unterteilen. Ein passiver Angreifer ist ein Beobachter. Er spioniert Informationen aus und zerstört dabei die Vertraulichkeit der Daten. Ein aktiver Angreifer begnügt sich nicht nur mit Beobachtungen, sondern greift aktiv in das Geschehen ein; Daten werden hinzufügt, gelöscht oder verändert. Ein aktiver Angriff zerstört nicht nur die Vertraulichkeit der Daten, sondern auch ihre Integrität und Authentizität. [MOV97]

Kommunikation und die dabei verwendeten Protokolle sind das Um und Auf eines mobilen Agenten: Kommunikation mit der Plattform, Kommunikation mit anderen Agenten, aber auch Kommunikation zwischen zwei Plattformen, insbesondere zum Zweck der Migration des Agenten. Zu den in [MOV97] aufgezählten klassischen Angriffen auf Protokolle zählen:

Angriff mit bekanntem Schlüssel (*Known-Key Attack*)

Bei der *Known-Key*-Attacke hat es der Angreifer bereits geschafft, in Besitz eines Schlüssels zu gelangen. Ausgehend von diesem Schlüssel versucht er nun, weitere Schlüssel davon abzuleiten. Ein Beispiel: Protokolle werden häufig mit Einmalschlüssel (*Session Keys*) gesichert, d. h. jede Gesprächseinheit (*Session*) wird mit eigens für diese Einheit erzeugten Schlüsseln gesichert. Ist die Gesprächseinheit beendet, werden die Schlüssel wieder verworfen. Diese Einmalschlüssel werden von einem oder mehreren Generalschlüsseln (*Master Keys*) abgeleitet. Bei einer *Known-Key*-Attacke kann der Angreifer nun mit Hilfe seines bereits abgefangenen Schlüssels versuchen, dieses Ableitungsverfahren zu knacken und so in Besitz weiterer oder zukünftiger Einmalschlüssel zu gelangen. Daher ist es wichtig, bei der Auswahl des Schlüsselableitungsverfahrens darauf zu achten, dass nur Algorithmen zur Anwendung kommen, die eine derartige Attacke verhindern (*One-Way Function*).

Angriff durch Wiedereinspielung *(Replay Attack)*

Bei dieser Art der Attacke zeichnet der Angreifer ein „Gespräch“ auf, um dieses zu einem späteren Zeitpunkt teilweise oder als Ganzes wieder abzuspielen. Ein Beispiel: A will sich gegenüber B authentifizieren. Beide Gesprächspartner sind in Besitz desselben geheimen symmetrischen Schlüssels K. A schickt nun seine mit K verschlüsselte ID an B, $enc<K>(ID_A)$. B entschlüsselt diese und erkennt darin die ID von A. Der Spion S, der diese Unterhaltung aufgezeichnet hat, sendet zu Beginn seines Angriffs ebenfalls $enc<K>(ID_A)$ an B, B entschlüsselt die Daten und identifiziert wiederum A als seinen Gesprächspartner. Der Spion S konnte somit, obwohl er nicht im Besitz des geheimen Schlüssels K ist, vortäuschen A zu sein.

Replay-Angriffe können durch die Verwendung von Zufallszahlen oder Sequenznummern verhindert werden. Für obiges Beispiel wäre die Lösung wie folgt: B übermittelt A zu Beginn des Gesprächs die Zufallszahl RND_B. A verschlüsselt nun nicht mehr nur seine ID_A, sondern auch die eben erhaltene Zufallszahl RND_B, $enc<K>(ID_A + RND_B)$, und schickt das Ergebnis an B. B entschlüsselt diese Da-

ten und verifiziert zuerst, ob RND_B wirklich der aktuellen Zufallszahl entspricht. Ist dies der Fall, kann B der ID von A vertrauen. Da der Spion S nicht in Besitz von K ist und B vor jeder Authentifizierung eine neue Zufallszahl aussendet, schlagen *Replay*-Attacken nun fehl. [RP-WIKI]

Identtiätswechsel (*Impersonation*)

Ein *Impersonation*-Angriff ist ein Angriff auf den Authentifikationsmechanismus eines Protokolls. Dabei versteckt sich der Angreifer hinter einer fremden Identität. Das zuvor erwähnte Beispiel, bei dem sich der Spion S als A ausgibt, kann auch als Beispiel für einen *Impersonation*-Angriff herangezogen werden.

Wörterbuchangriff *(Dictionary Attack)*

Der *Dictionary*-Angriff ist ein typischer Angriff auf Passwörter. Üblicherweise wird über Passwörter ein Hash berechnet und das Ergebnis dieser Berechnung abgespeichert. Die Verifizierung während der Passworteingabe erfolgt durch Vergleich der Hashwerte. Ein Angreifer geht nun von der Annahme aus, dass das gesuchte Passwort ein reales Wort ist und arbeitet eine Liste von Wörtern ab, deren Hashwert er mit dem des abgespeicherten Passwortes vergleicht. „Bei einem aktiven Wortschatz von 50.000 Wörtern pro Sprache können selbst auf handelsüblichen Rechnern dutzende Sprachen innerhalb weniger Sekunden ausprobiert werden. Ein einzelnes Wort als Schlüssel ist daher sehr unsicher.“ [WB-WIKI] Dies ist der Grund dafür, dass immer mehr Programme bei Passwörtern zwingend die Verwendung von Sonderzeichen vorschreiben.

Forward Search Attack

Der *Forward-Search*-Angriff ist ein naher Verwandter des Wörterbuchangriffs. Er richtet sich aber nicht gegen Passwörter, sondern sein Ziel ist die Entschlüsselung von Nachrichten. Kennt ein Angreifer Länge und Inhalt eines bestimmten Feldes, z. B. den bezahlten Betrag innerhalb einer elektronischen Zahlungstransaktion, kann er, ausgehend von den maximal möglichen Eingangsdaten, versuchen den Verschlüsselungsalgorithmus herauszufinden. Bei einem 32 Bit Feld gibt es 2^{32} mögliche Eingangsdaten. Würde der Angreifer Algorithmus und Schlüssel bereits kennen, müsste er maximal 2^{32} Berechnungen durchführen und seine Ergebnisse jeweils mit dem Inhalt des gesuchten Feldes vergleichen. Sobald die beiden Werte übereinstimmen, kennt er, auf Grund der von ihm verwendeten Eingangsdaten, den tatsächlich verschlüsselten Betrag.

Verschachtelter Angriff *(Interleaving Attack)*

Ein *Interleaving*-Angriff geht meistens Hand in Hand mit einem *Impersonation*-Angriff während einer Authentifizierung. Ein Angreifer C baut z. B. mit zwei Komponenten A und B eine Verbindung auf und lässt beide glauben miteinander zu kommunizieren, obwohl die stattfindende Kommunikation in Wahrheit über C abläuft. In diesem Fall kann C in die Kommunikation zwischen A und B eingreifen, indem er z. B. Nachrichten abfängt und seine eigenen Nachrichten in das Protokoll einbringt.

3.4 Bewertung von Sicherheit

Der Wunsch, Sicherheit mess- und bewertbar zu machen, führte im Laufe der Zeit zur Entwicklung verschiedener regionsspezifischer Standards. In Europa wurde ITSEC, Information Technology Security Evaluation Criteria, entwickelt, in Amerika TCSES, Trusted Computer System Evaluation Criteria. Mit CC, Common Criteria for Information Technology Security Evaluation, wurde ein internationaler Standard geschaffen, der die beiden oben erwähnten ablösen soll. Für Interessierte sei hier die Internetseite des deutschen Bundesamts für Informationstechnologie erwähnt (http://www.bsi.bund.de/cc/index.htm).

Zur Bestimmung der Sicherheit von kryptographischen Primitiven gibt es verschiedene Modelle [MOV97]:

Perfekte Sicherheit *(Unconditional Security)*

Unconditional Security ist das, was sich jeder für sein sicherheitskritisches System wünscht, die perfekte Sicherheit. Obwohl ein Angreifer unbegrenzte Ressourcen zur Verfügung hat, wird sein Angriff dennoch ohne Erfolg bleiben. Diese Stufe an Sicherheit wird kaum je erreicht, *One Time Pads* gehören hier zu den seltenen Ausnahmen.

One Time Pads gehören zu den symmetrischen Verschlüsselungsverfahren. Für symmetrische Verschlüsselungsverfahren ist eine Grundvoraussetzung zur Erlangung von perfekter Sicherheit, dass der Schlüssel genauso lang wie die zu verschlüsselnde Nachricht sein muss und nur ein einziges Mal verwendet werden darf. Ein Beispiel eines *One Time Pads* aus [OTP-WIKI]: Empfänger und Sender sind beide in Besitz einer zufälligen Reihenfolge an Buchstaben, dem Schlüssel. Zur Verschlüsselung benötigt der Sender die gleiche Anzahl an Buchstaben aus dem Schlüssel wie die Nachricht hat. Schlüssel und Nachricht werden nun Buchstabe für Buchstabe miteinander verknüpft. Man wandelt z. B. den Buchstaben A in 1 um, was seiner Position im Alphabet entspricht. Die entsprechenden Werte der Buchstaben aus Schlüssel und Nachricht werden nun miteinander addiert und Modulo 26 gerechnet, da das Alphabet nur 26 Buchstaben hat und die Ergebnisse wiederum in Buchstaben dargestellt werden.

Schlüssel:	W Z S L X W M F Q U D M P J L Y Q O X X B
Nachricht:	A N G R I F F I M M O R G E N G R A U E N
Verschlüsselung:	X N Z D G C S O D H S E W O Z F I P S C P

Durch Subtraktion des Schlüssels vom Geheimtext gelangt man wieder zur ursprünglichen Nachricht.

One Time Pads sind nur *unconditional secure*, wenn ihr Schlüssel geheim bleibt und kein Schlüssel mehrmals verwendet wird. Asymmetrische Verschlüsselungsverfahren können nicht *unconditional secure* sein, da man, wenn das verschlüsselte Ergebnis C bekannt ist, die Eingangsdaten dadurch ermitteln kann, dass man alle möglichen Eingangsdaten solange verschlüsselt bis man das Ergebnis C erhält.

Beweisbare Sicherheit (*Provable Security*)

Hier wird die Schwierigkeit eines Angriffs mit der Schwierigkeit zur Findung einer Lösung für ein bekanntes mathematisches Problem gleichgesetzt, etwa dem, dass z. B. die Faktorisierung bestimmter großer ganzer Zahlen praktisch unmöglich ist. Man reduziert das eigentliche Problem auf ein mathematisch bereits gut untersuchtes Problem. Dies ist somit kein direkter Beweis, da hier zwei Probleme miteinander in Relation gesetzt werden. *Provable Security* wird oft als Untergruppe der nachfolgend beschriebenen *Computational Security* gesehen.

Effiziente Sicherheit (*Computational Security*)

Effiziente Sicherheit stellt den Aufwand, der benötigt wird ein System erfolgreich zu verteidigen, dem Aufwand, den ein Angreifer für eine erfolgreiche Attacke benötigt, gegenüber. Dabei wird davon ausgegangen, dass dem Angreifer die bestmöglichen Methoden und Ressourcen zur Verfügung stehen. Der von beiden Seiten benötigte Aufwand wird gemessen. Übersteigt der Aufwand des Angreifers den Aufwand des Systems deutlich, spricht man von effizienter Sicherheit. Voraussetzung ist, dass das System zuvor ausreichend analysiert wurde, um die relevanten Angriffsszenarien ausfindig zu machen.

Wie bei der beweisbaren Sicherheit wird das untersuchte Problem auf ein anderes Problem reduziert, allerdings fehlt im Falle der effizienten Sicherheit der Beweis dafür, dass die beiden Probleme zueinander auch wirklich äquivalent sind. Ein Großteil der verbreitetsten asymmetrischen und symmetrischen Primitive fallen in diese Kategorie. Effiziente Sicherheit wird im Englischen auch häufig als *Practical Security* bezeichnet.

Ad-hoc-Sicherheit

Unter dem Begriff der Ad- hoc-Sicherheit fallen alle überzeugend dargebrachte Argumente, die zeigen, dass die für einen erfolgreichen Angriff benötigten Ressourcen (wie z. B. Zeit) die vorhandenen Ressourcen eines Angreifers weit übersteigen. Kryptographische Primitive und Protokolle, die einer derartigen Analyse standhalten, liefern eine heuristische Sicherheit. Ad-hoc-Sicherheit ist wohl das am meisten verbreitete Modell, sollte aber trotzdem hinterfragt werden. Die meisten Primitive und Protokolle wurden zur Abwehr klassischer Angriffsszenarien entwickelt (siehe Abschnitt 3.3), unvorhergesehene Angriffe stellen somit weiterhin eine Gefahr dar.

Das Vertrauen in effiziente und Ad-hoc-Sicherheit steigt mit dem Grad an Aufwand und Zeit, der in die Analyse des Problems investiert wurde. Aber Zeit und Aufwand alleine reichen nicht, wenn nur einige wenige mit dieser Aufgabe betraut sind. Je mehr Personen in die Analyse involviert sind, desto mehr Vertrauen kann dem Resultat entgegen gebracht werden.

3.5 Sicherheit und Rechtsverbindlichkeit

Im Abschnitt 3.1.3 wurden digitale Signaturen vorgestellt. Mit Hilfe dieser Signaturen können sich mobile Agenten gegenüber ihren Wirtsplattformen oder anderen Agenten authentifizieren und gege-

benenfalls Geschäfte abwickeln. Doch welche rechtlichen, aber technisch umzusetzenden, Voraussetzungen muss ein Agent erfüllen, um auch in den Augen des Gesetzes für seinen Eigentümer rechtswirksam handeln zu dürfen?[1]

Das deutsche Signaturgesetz unterscheidet drei Klassen elektronischer Signaturen, jede Klasse entspricht einer bestimmten Qualitätsstufe. „Je höherwertiger die Signatur, desto mehr Bedeutung hat sie für den Rechtsverkehr, und desto größer ist ihre Funktionalität.“ [SIG-WIKI]

In [BNS05] werden die drei Signaturklassen vorgestellt und bezüglich ihrer rechtlichen Einsatzfähigkeit in (mobilen) Agenten analysiert:

3.5.1 Einfache elektronische Signatur

Hier handelt es sich um elektronische Daten, die mit anderen elektronischen Daten verknüpft sind und zur Authentifizierung dienen. Kryptographie ist für die einfache elektronische Signatur nicht erforderlich. Wird die handschriftliche Unterschrift eines Agenteneigentümers eingescannt und in ein elektronisches Dokument eingefügt, so ist dies eine „einfache elektronische Signatur“. Einfache elektronische Signaturen können für formfreie Vereinbarungen eingesetzt werden [SIG-WIKI]. Agenten sind somit durchaus in der Lage diese Art von Signatur zu erzeugen bzw. zu verwenden.

3.5.2 Fortgeschrittene elektronische Signatur

Die fortgeschrittene elektronische Signatur muss zusätzlich noch folgende vier Anforderungen erfüllen:

1. Sie ist ausschließlich dem Signaturschlüsselinhaber zugeordnet.
   Als Signaturschlüsselinhaber wird hier eine natürliche Person angesehen, im Falle eines mobilen Agenten ist dies der Eigentümer des Agenten. Die ausschließliche Zuordnung geschieht bei einem asymmetrischen Verfahren über den privaten Schlüssel. Diese Zuordnung bleibt auch bestehen, wenn nicht nur ein, sondern mehrere Agenten den Schlüssel ihres Eigentümers einsetzen. Die Existenz eines Zertifizierungsdienstes ist bei der fortgeschrittenen elektronischen Signatur nicht erforderlich.
2. Sie ermöglicht die Identifizierung des Signaturschlüsselinhabers.
   In der Regel geschieht diese Identifizierung mit Hilfe eines Zertifikates. An die Art der Identifikation bei der Ausstellung des Zertifikates werden keine besonderen Anforderungen gestellt.
3. Sie wird mit Mitteln erzeugt, die der Signaturschlüsselinhaber unter seiner alleinigen Kontrolle hält.

[1] Der Abschnitt „Sicherheit und Rechtsverbindlichkeit“ stützt sich auf Artikel, die sich auf deutsches, nicht österreichisches, Recht beziehen. Eventuellen Unterschieden zum österreichischen Recht wurde nicht explizit nachgegangen. Beide stützen sich aber auf Richtlinien des europäischen Parlamentes und des Rates über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen. Siehe [SIG-AT], [SIG-DE].

Wird ein asymmetrisches Verfahren verwendet, versteht man unter alleiniger Kontrolle, dass der private Schlüssel des Agenteneigentümers vor dem Zugriff unbefugter Personen geschützt werden soll. Bei der fortgeschrittenen elektronischen Signatur wird dabei nicht differenziert, ob dieser Schlüssel lediglich durch ein Passwort geschützt auf der Festplatte gespeichert wird oder ob der Zugriff auf ihn nur über eine eigene Kryptohardware möglich ist.

4. Sie ist mit den Daten, auf die sie sich bezieht, so verknüpft, dass eine nachträgliche Veränderung der Daten erkennbar ist.
   Dies kann durch die bereits im Abschnitt 3.1.2 vorgestellten Hashfunktionen geschehen.

Im Falle eines Rechtsstreit werden fortgeschrittene elektronische Signaturen wie einfache elektronische Signaturen als Objekte des Augenscheins behandelt, d. h. die sich auf die Signatur beziehende Partei muss beweisen, dass digitale Signatur und Identifizierungsmerkmal echt sind. Fortgeschrittene elektronische Signaturen können für formfreie Vereinbarungen eingesetzt werden. [SIG-WIKI]

Beispiele für fortgeschrittene elektronische Signaturen sind PGP (*Pretty Good Privacy*) oder GPG (*GNU Privacy Guard*). Bei beiden Programmen kann der Benutzer sich sein eigenes Schlüsselpaar erzeugen. Der private Schlüssel wird dabei passwortgeschützt auf der Festplatte abgespeichert, mit ihm kann der Benutzer nun beliebige Daten signieren. Diese Art von elektronischer Signatur lässt sich problemlos erstellen und ist somit auch für mobile Agenten leicht nutzbar.

3.5.3 Qualifizierte elektronische Signatur

Bei der nächsthöheren Klasse, der qualifizierten elektronischen Signatur, kommen wiederum zwei Anforderungen hinzu:

1. Sie muss auf einem qualifizierten Zertifikat beruhen, dass zum Zeitpunkt der Erstellung der Signatur auch gültig war.
   Die Beantragung eines qualifizierten Zertifikates ist ein, zumindest für einen längeren Zeitraum, einmaliger Prozess. Einmal erhalten, können auch mobile Agenten auf diese Zertifikate zurückgreifen.

2. Sie muss von einer sicheren Signaturerstellungsbehörde erstellt werden.
   Laut Gesetz erfordert eine sichere Signaturerstellungsbehörde die Identifikation des Schlüsselinhabers durch „Besitzt und Wissen“, d. h. er muss z. B. eine Chipkarte „besitzen“ und die dazugehörige „Geheimnummer“ wissen.

Die qualifizierte elektronische Signatur ist der eigenhändigen Unterschrift weitgehend gleichgestellt. „Bis auf einige Ausnahmen wie z. B. Bürgschaften, Kündigung von Arbeitsverträgen und Zeugnisse können Dokumente und Verträge elektronisch signiert werden und sind auch vor Gericht als Beweismittel anerkannt. Dies führt zu einer Beweislastumkehr. Bei einem qualifiziert elektronisch signierten Dokument muss im Zweifelsfall die Ungültigkeit der Signatur bewiesen werden.“ [SIG-QES]

Es sieht so aus als würde Punkt 2 den Einsatz einer qualifizierten elektronischen Signatur in Agenten verhindern, denn müsste der Eigentümer jeden Signaturvorgang durch seine Identifikation bestätigen, wäre das Handeln des Agenten nicht mehr autonom. Ein Ausweg liefert aber die dem Paragra-

phen beiliegende amtliche Begründung. Hier wird davon ausgegangen, dass die erforderliche Identifikation einmal durchgeführt wird und dann für eine gewisse Zeit bzw. eine gewisse Anzahl von Signaturvorgängen gültig ist. Allerdings muss sichergestellt sein, dass der Agent sich noch im administrativen Umfeld seines Eigentümers befindet. Hinter der Aufforderung zur Bestätigung versteckt sich aber auch eine Gefahr, denn es ist möglich, dass dem Signierenden zur Bestätigung andere Daten, als die später tatsächlich signierten Daten, angezeigt werden. Werden mehrere Signaturvorgänge erlaubt, kann es geschehen, dass der Signierende überhaupt nicht bemerkt, dass Signaturen in seinem Namen ausgestellt werden.

Mobile Agenten werden auf Plattformen ausgeführt, die nicht unter der administrativen Kontrolle des Agenteneigentümers stehen. Es besteht ein hohes Risiko, dass der mobile Agent auf seiner Reise durch das Netz manipuliert wurde und nicht mehr die eigentlichen Ziele seines Eigentümers verfolgt. In diesem Fall können die Sicherheitsanforderungen der qualifizierten elektronischen Signatur nicht mehr erfüllt werden, da der Anscheinsbeweis nicht erbracht werden kann. Der Anscheinsbeweis zieht Schlüsse von bewiesenen auf zu beweisende Tatsachen, etwa zur Feststellung von Kausalität und Verschulden. [AB-WIKI]

3.5.4 Attributzertifikat

Vom Gesetz her gibt es keine Beschränkung über die maximale Höhe eines Geschäftsabschlusses, den ein Agent im Auftrag seines Eigentümers durchführen darf. Zur Absicherung aller wäre aber die Festlegung eines Höchstbetrages durchaus sinnvoll. Das Signaturgesetz bietet die Möglichkeit an, qualifizierte elektronische Signaturen mit Attributen zu versehen. Diese Attribute enthalten Informationen, die die „Vertretungsmacht“ der Signatur genauer definieren, z. B. eben ein maximales Transaktionsvolumen festlegen können. Ein Benutzer kann sein qualifiziertes Zertifikat aber nicht selbst mit Attributen versehen, laut Gesetz darf dies nur ein Zertifizierungsdienstanbieter, der besondere Voraussetzungen erfüllen muss.

Es gibt aber auch reine Attributzertifikate, die von einer vertrauenswürdigen Zertifizierungsstelle ausgestellt werden. Attributzertifikate binden die in ihnen enthaltenen Attribute elektronisch an ein Identitätszertifikat. Auf diese Weise können ein und demselben Zertifikat je nach Anwendungsfall unterschiedliche Attribute zugeordnet werden, ohne dass das Originalzertifikat geändert werden muss. Die Zertifizierungsstelle des Attributzertifikats muss nicht mit der Zertifizierungsstelle der elektronischen Signatur identisch sein. [AZ-WIKI].

Rechtlich gesehen ist ein Agent kein Stellvertreter sondern lediglich ein Hilfsmittel. Stationäre Agenten können im Auftrag ihrer Benutzer Verträge abschließen und grundsätzlich alle Signaturstufen verwenden. Für mobile Agenten gilt dies ebenso, allerdings mit der Einschränkung, dass sie keine qualifizierte elektronische Signatur verwenden können, da sie die dazu benötigten rechtlichen Voraussetzungen nicht erfüllen. Das Risiko eines Fehlverhaltens des Agenten kann durch Attributzertifikate begrenzt werden.

Der vorangegangene Abschnitt hat sich mit der Frage beschäftigt, welche Voraussetzung ein Agent erfüllen muss, um im Namen seines Eigentümers elektronische Geschäfte abwickeln zu können. Dabei darf aber nicht darauf vergessen werden, dass der (mobile) Agent gleichzeitig auch Vorkeh-

rungen treffen muss, um im Falle einer gerichtlichen Streitfrage, etwa wenn eine Seite die Existenz des getätigten Vertrages leugnet, Unterlagen zur Beweisführung beibringen zu können. Entsprechende gesicherte Traces und Logs sind vorzusehen.

3.6 Transparente Kommunikation und sichere Namensgebung

Das Besondere an einem mobilen Agenten ist, wie der Name schon sagt, seine Mobilität. Da Agenten per Definition aber auch autonom agieren lässt sich nicht vorhersagen, wann sie sich wo aufhalten werden. Mobile Agenten können vor Nachrichten „davonlaufen“. Dies ist ein besonderer Aspekt bei der Kommunikation zwischen Agenten zweier Plattformen. Agent A will zwar mit Agent B kommunizieren, weiß aber nicht wo Agent B sich zurzeit aufhält. Diese Art der Kommunikation nennt sich transparente Kommunikation, transparent deshalb, weil Agent A nicht weiß, wo sich sein gewünschter Gesprächspartner Agent B aufhält. [R01c]

Transparente Kommunikation teilt sich in zwei Aufgaben:

- Finden des Agenten
- Übermitteln der Nachricht

Damit Agenten plattformübergreifen miteinander kommunizieren können müssen ihre Wirtsplattformen eine entsprechende Infrastruktur anbieten. Für das Finden der Agenten gibt es unterschiedliche Ansätze, etwa der, dass jeder Agent bei seiner Migration allen Plattformen des Systems mitteilt wohin er migriert. Dies ist allerdings ein sehr kostenintensiver Ansatz, da es hier zu einer Vielzahl an Nachrichten kommt und Informationen redundant gespeichert werden, da jede Plattform dasselbe Wissen über alle Agenten speichert. Daher gibt es unter anderem auch den Ansatz, diese Informationen auf einer zentralen Plattform zu speichern. [BR05]

Eines ist aber allen Ansätzen gemeinsam; sie benötigen eine Form von Namensdienst, der die eindeutige Kennung eines Agenten auf seine aktuell bekannte Position abbildet. Befinden sich die Agenten nicht in einem abgeschlossenen System, sondern z. B. im Internet, stellt dies an den Namensdienst eine hohe Anforderung. Ein zentralisierter Namensdienst ist hier kaum zu realisieren. Bei einem verteilten Namensdienst muss der für den Agenten zuständige Namensdienst aus dem Namen des Agenten hervorgehen. Diese Information muss vom Eigentümer des Agenten bestätigt und von den Plattformen überprüfbar sein, z. B. durch Verwendung einer digitalen Signatur. Anderenfalls könnten Angreifer den Namen und dadurch auch den Namensdienst des Agenten fälschen; dies könnte z. B. dazu führen, dass der Agent nicht mehr ausgeführt wird (*Denial Of Service*).

Durch diese Vorgangsweise gelangen Namensdienste aber an viel Wissen, da sie die Routen, der von ihnen verwalteten Agenten nachverfolgen und mit Hilfe dieser Information u. U. auch vorausberechnen können. Die Privatsphäre der Eigentümer der Agenten wird dadurch aufgeweicht.

Ein anderer Ansatz ist daher die implizite Namensgebung. In [R01c] wird davon gesprochen jeden Agenten mit einem statischen Kern zu versehen und diesen vom Eigentümer zu signieren. Dieser, während des ganzen Lebenszyklus des Agenten unveränderliche Kern (dazu zählt z. B. der Programmcode), dient als Anker zur Sicherung weiterer Informationen. Um die Eindeutigkeit dieser Sig-

natur noch zu erhöhen, sollte man diesen statischen Daten vor der Berechnung der Signatur noch Zufallsdaten hinzufügen, diese können z. B. Datum und Uhrzeit sein. Der Name des Agenten wird nun implizit aus einer Hashfunktion und den Signaturdaten gewonnen.

Diese Art der Namensgebung bringt einige Vorteile mit sich:

- Der Name des Agenten kann nicht erraten oder gefälscht werden, es sei denn der private Signaturschlüssel gerät in falsche Hände.
- Gehören mehrere unterschiedliche Agenten einem Eigentümer, so ist diese Zuordnung nicht mehr aus dem Namen ersichtlich.
- Der Namensdienst kann aus dem Namen des Agenten nicht mehr auf dessen Eigentümer rückschließen, es sei denn er fragt explizit bei der Wirtsplattform des Agenten nach.
- Agenten können die Wirtsplattformen hinsichtlich ihres Namens nicht mehr täuschen, da die Plattform den Namen selbst berechnet. Ändert sich der statische Kern eines Agenten oder sein Eigentümer, ändert sich dadurch auch seine Signatur und somit auch der daraus berechnete Hashwert, d. h. der implizite Name des Agenten.
- Die vom Namensdienst zu verwaltende Datenmenge verringert sich, da Hashwerte im Allgemeinen wesentlich kürzer sind als Signaturen.

3.7 Standards bei (mobilen) Agenten

Die technische Voraussetzung für die Interaktion unterschiedlichster Agenten und Plattformen ist eine gemeinsame Struktur. Häufig wird dies nur dadurch sichergestellt, dass die Agentenplattformen vom gleichen Hersteller stammen und die Agenten mit dem gleichen Toolkit erstellt wurden. Für einen breiten Einsatz ist eine Standardisierung der Infrastruktur und Kommunikation unerlässlich. Standards ermöglichen Herstellerunabhängigkeit und helfen Kosten zu reduzieren, allein schon dadurch, dass nicht jede Neuentwicklung von Null an beginnen muss sondern auf bereits Vorhandenem aufbauen kann. [PPW02]

Die vorhandenen Standardisierungsansätze rund um Sicherheit in „mobilen“ Agentensystemen decken entweder nur einen kleinen Bereich ab oder wurden bis dato nicht weiterverfolgt. Ein Besuch der Internetseiten der beiden im Folgenden vorgestellten Standards für (mobile) Agenten, FIPA und MASIF, erweckt den Anschein, als wäre die „Hohe Zeit“ der Agenten vorbei; die Seiten wirken teils ungewartet und Links verlaufen ins Leere. Die aktuellen Ergebnisse in Forschung und Entwicklung, die auf diesen Standards aufbauen, finden sich in separat publizierten wissenschaftlichen Artikeln wieder, zumeist unabhängig von den oben erwähnten Instituten.

3.7.1 NIST

NIST steht für „*National Institute of Standards and Technology*“ und ist eine Bundesbehörde der Vereinigten Staaten mit Hauptsitz in Gaithersburg, Maryland. Das Institut untersteht dem „*United States Department of Commerce*“ und hat die Aufgabe, die amerikanische Wirtschaft und deren industriellen Wettbewerb durch das Entwickeln und Festlegen von Standards und wissenschaftlicher Messgrößen zu stärken. Unter anderem ist NIST Herausgeber der FIPS Standards (*Federal Information Processing Standards*). Diese Standards sind für die amerikanischen Bundesbehörden, je nachdem wie sie vom *Secretary of Commerce* eingestuft wurden, entweder Richtlinien oder auch zwingende Vorgaben. So wird z. B. die Verwendung des AES Standards (FIPS PUB 197) zur Verschlüsselung sensitiver Daten zwingend vorgegeben [NIST Homepage].

Im Oktober 1999 wurde von NIST eine Publikation der 800er Serie zum Thema „Mobile Agent Security“ (NIST 800-19) [JK99] veröffentlicht. Die Publikationen der 800er Serie sind keine Standards, sondern enthalten Dokumente allgemeinen Interesses zum Thema Sicherheit und Computer. Die 800er Serie berichtet über Erfahrungen und Richtlinien zum Thema Sicherheit und Computer und bemüht sich um die Zusammenarbeit zwischen Industrie, Regierung und akademischen Einrichtungen. Die Publikation NIST 800-19 zum Thema „Mobile Agent Security“ wird, auch heute noch, in der Literatur gern zitiert und soll daher, auch wenn sie kein Standard ist, nicht unerwähnt bleiben.

Das Dokument richtet sich an Designer und Entwickler von Agentensystemen und behandelt folgende Themen:

- Aufzeigen möglicher Bedrohungsszenarien
- Aufzeigen allgemeiner Sicherheitsziele
- Vorstellen von Methoden, den Bedrohungen entgegen zu wirken, bzw. die Sicherheitsziele zu erreichen

Die in [JK99] vorgestellten Richtlinien sind sozusagen *State Of The Art* in der Entwicklung von Agentensystemen und finden sich auch im Kapitel „4 Sicherheitsanalyse von Agentensystemen“ und im Kapitel „6 Umsetzung der Sicherheitsanforderungen“ wieder.

3.7.2 FIPA

Die *Foundation for Intelligent Physical Agents*, kurz FIPA genannt, wurde 1996 von einer Schweizer Firma ins Leben gerufen. Ihr Ziel war es, standardisierte Spezifikationen für heterogene, miteinander interagierende Agenten und Agentensysteme zu schaffen. Im Jahr 2005 wurde FIPA zum elften Mitglied des Standard Komitees der IEEE Computer Society [FIPA].

Die von FIPA veröffentlichten Dokumente unterliegen einem Lebenszyklus: *Preliminary*, *Experimental*, *Standard*, *Deprecated* und *Obsolete*. *Preliminary* ist eine vorläufige Spezifikation. *Experimental* ist eine von FIPA akzeptierte Spezifikation, zu der es aber noch keine Implementierungen gibt. Gibt es mehr als eine Implementierung der Spezifikation, bekommt sie den Status *Standard*. Spezifikationen, die nicht mehr gültig sind, erhalten etwa ein halbes Jahr lang des Status *Deprecated*, der dann in *Obsolete* übergeht [PPW02].

Aktuell (September 2008) finden sich keine *Preliminary*, *Experimental* oder *Standard* Spezifikationen auf der FIPA Homepage, die sich explizit dem Thema mobile Agenten oder Sicherheit widmen. Im Standard SC00001L „*FIPA Abstract Architecture Specification*“ [FIPA02a] findet sich ein sehr kurzer Anhang mit dem Titel “*Annex D - Goals for Security and Identity Abstractions*”, in dem u. a. eine kurze Übersicht zu Sicherheitsanforderungen wie Authentifizierung, Vertraulichkeit, Integrität sowie Zugangskontrollen gegeben wird, Vorschläge gemacht werden, in welchen Teilen des FIPA-Modells diese Sicherheitsmaßnahmen umgesetzt werden könnten und eine Reihe von Risiken aufgezählt werden, die von FIPA nicht behandelt werden, da sie nicht agentenspezifisch sind und Software im allgemeinen betreffen. Zu diesen Risiken zählen u. a. das Ausspionieren und Verändern von Daten, *Cut-And-Paste-* und *Denial-Of-Service-*Attacken oder böswillig kooperierende Agenten (siehe Kapitel 4). Zum Thema Sicherheit gibt eine obsolete Spezifikation OC00020 mit dem Titel „*FIPA 98 Part 10 Version 1.0: Agent Security Management Specification*“ [FIPA98], deren Ziel es war, die Anforderungen an eine sichere Kommunikation, sowohl innerhalb einer Plattform als auch zwischen mehreren Plattformen, aufzuzeigen. Zur Überwachung der Plattform und ihrer Sicherheitsrichtlinien wurde hier ein *Agent Platform Security Manager* (APSM) eingeführt, der die Kommunikation zwischen Agent und Plattform überwachen sollte. Zum Thema mobile Agenten gibt es eine *deprecated* Spezifikation DC00087 mit dem Titel „*FIPA Agent Management Support for Mobility Specification“* [FIPA02c], deren Ziel es war, die grundlegendsten Anforderungen und Technologien mobiler Agenten zu definieren.

Allerdings finden sich Hinweise auf Aktivitäten zu den Themen mobile Agenten oder Sicherheit, etwa die *Mobile Agent Working Group* (http://www.fipa.org/subgroups/MA-WG.html), die für Juni 2006 Ergebnisse versprach[2] und das *Security Technical Committee* (http://www.fipa.org/activities/security.html) mit einem finalen Meilenstein für Februar 2002[3]. [FIPA]

Aufbau des FIPA-Modells

Der zentrale Bestandteil des FIPA-Modells ist die Plattform, sie stellt die Basisinfrastruktur zur Verfügung. Drei wesentliche Dienste der Agentenplattform sind das *Agent Management System* (AMS), der *Directory Facilitator* (DF) und das *Message Transport Service* (MTS). Diese drei Dienste bilden die *Agent Management Services* [FIPA04] (siehe Abbildung 3.3).

Das AMS stellt Funktionen zum Erzeugen und Löschen von Agenten zur Verfügung. Es enthält auch ein Namensverzeichnis aller auf der Plattform verfügbaren Agenten (*White Pages*). Das DF funktioniert wie die gelben Seiten des Telefonbuches (*Yellow Pages*). Agenten, die Dienste anbieten, registrieren diese beim DF. Agenten, die Dienste benötigen, schlagen diese im DF nach. Das *Message Transport Service* stellt einen Mechanismus für den Austausch von Nachrichten zur Verfügung.

[2] Auf Rückfrage erhielt ich die Auskunft, dass die *Mobile Agent Working Group* zwar nicht aufgelöst, aber zurzeit auch nicht aktiv ist.

[3] Eine Anfrage zum aktuellen Stand an den Vorsitzenden des *Security Technical Committee* war leider nicht erfolgreich, da die angegebene eMail-Adresse nicht mehr gültig zu sein scheint, gleiches gilt für die auf der Internet-Seite angegebene externe Homepage.

Agenten auf derselben Plattform verwenden einen herstellerspezifischen Nachrichtendienst, der von FIPA nicht näher spezifiziert wird. Agenten auf unterschiedlichen Plattformen kommunizieren über einen *Agent Communication Channel* (ACC), der aber auch Teil des MTS ist.

Zur Kommunikation zwischen den Agenten wird eine *Agent Communication Language* (ACL) verwendet. In der ACL läuft die Kommunikation über Sprechakte ab, ein Sprecher sagt durch das Sprechen nicht nur etwas aus, sondern führt damit gleichzeitig eine Aktion aus. So kann z. B. Agent A über das Schlüsselwort „*request*“ eine Anfrage zur Ausführung an Agent B schicken, etwa eine Anfrage ein bestimmtes File, das in den mitgeschickten Daten genauer bezeichnet wird, für das Schreiben zu öffnen. [PPW02], [FIPA02b]

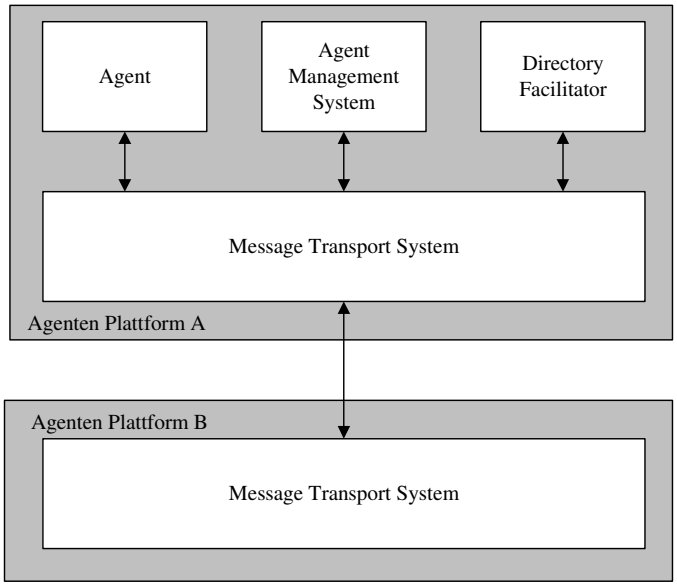


Abbildung 3.3: ***FIPA Agent Management Referenz Model*** **[FIPA04]**

In [OC07] wurde das FIPA-Modell bezüglich seiner Anfälligkeit gegen Angriffe analysiert und an Hand des Agentensystems Victor zu den nachfolgend beschriebenen Sicherheitslücken auch Lösungsansätze vorgestellt.

Sicherheitslücken des *Directory Facilitator*

Agenten, die am AMS registriert sind, können dem DF Beschreibungen zu den Diensten, die von ihnen zur Verfügung gestellt werden, übermitteln. Laut der Spezifikation für das DF müssen sich die Agenten dazu nicht authentifizieren, auch gibt es keine Längenbegrenzungen zu den Beschreibungen der Dienste. Ein böswilliger Agent könnte eine extrem lange Dienstbeschreibung an den DF schicken, so dass dessen gesamter verfügbarer Speicher verbraucht wird und er dadurch die Dienste anderer Agenten nicht mehr eintragen kann. Auch kann ein böswilliger Agent durch dauerndes Schreiben oder Abfragen von Diensten den DF überlasten.

Sicherheitslücken des *Agent Management Systems*

Jeder Agent muss sich am AMS seiner Wirtsplattform registrieren. Der Agent teilt dem AMS auch seine Zustandswechsel mit. Bei dieser Kommunikation wird wiederum keine Authentifizierung gefordert. Ein böswilliger Agent kann das AMS durch das Absenden zu vieler unsinniger Zustandswechsel überlasten und so andere Agenten behindern.

Sicherheitslücken des *FIPA Trust Models*

Das aktuelle FIPA-Modell erlaubt es Agenten, ohne vorhergehende Authentifizierung oder Autorisierung, Aktionen zu setzen. Ein Agent B könnte daher die Dienstbeschreibungen eines anderen Agenten A im DF, oder dessen Zustände im AMS, böswillig verändern. Aber auch der Plattform selbst kann nicht vertraut werden, gelingt es einem böswilligen DF sich als gutwilliges DF auszugeben (Identitätswechsel), so kann dieser z. B. eintreffende Serviceanfragen an einen, mit ihm zusammenarbeitenden, böswilligen Agenten weiterleiten. Gleiches gilt auch für das AMS. Gibt sich ein böswilliger AMS als gutwilliger AMS aus, so kann dieser z. B. böswillige Agenten registrieren, die danach ihre Dienste dem DF mitteilen können.

Sicherheitslücken des *Message Transport Services*

Das MTS dient zur Kommunikation zwischen den Agenten. Ein böswilliger Agent kann ein MTS durch das Senden zu vieler Dienstanfragen oder durch das Senden von zu langen Nachrichten überlasten und so die Kommunikation zwischen den anderen Agenten beeinflussen.

Sicherheitslücken durch falsch aufgebaute Nachrichten

Sendet ein böswilliger Agent falsch aufgebaute Nachrichten, so führt dies im DF, AMS oder MTS zu einer Fehlerbehandlung. Zu viele dieser falsch aufgebauten Nachrichten können die einzelnen Module überlasten, z. B. Anforderungen von nicht existierenden Diensten an das DF, Anfragen an nicht existierende Agenten im MTS oder falsch aufgebaute Registrierungen an das AMS bzw. falsch aufgebaute Dienstbeschreibungen an das DF.

3.7.3 OMG MASIF

Die OMG, die *Object Management Group*, wurde 1989 gegründet und ist eine internationale Non-Profit Organisation. Sie zählt mehr als 800 Mitglieder und ist damit das weltgrößte Software Konsortium. Ihr Ziel ist es, Standards für verteilte objektorientierte Softwarearchitekturen zu schaffen. Ein bekannter OMG Standard ist die *Common Object Broker Request Architecture*, CORBA. CORBA ist eine Spezifikation, die plattformübergreifende Protokolle und Dienste definiert und das Erstellen verteilter Anwendungen in heterogenen Umgebungen vereinfacht [CORBA-WIKI].

1995 wurde von OMG ein *Request for Proposal* zum Thema mobile Agenten erstellt, der 1997 zur Herausgabe des MASIF-Standards führte. MASIF steht für *Mobile Agent System Interoperability Facility* [OMG97] und baut sehr stark auf CORBA auf. Die aktuelle Dokument Version aus dem Jahr 2000 wird von der OMG *Mobile Agent Facility Spezification* genannt [OMG00]. Auch bei

OMG findet sich eine Arbeitsgruppe „*OMG Agent Platform Special Interest Group*“ mit dem Ziel die Objekt Management Architektur (OMA)[4] der OMG so zu erweitern, dass eine gute Unterstützung der Agenten Technologie gegeben ist (http://www.objs.com/agent/; abgerufen im September 2008). Zu den Aufgabengebieten dieser Arbeitsgruppe zählen u. a. auch Sicherheit und mobile Agenten. Das letzte Meeting der Gruppe fand im Jahr 2002 statt. [PPW02], [DK06]

Da MASIF auf CORBA aufbaut, folgt eine kurze Beschreibung von CORBA.

CORBA

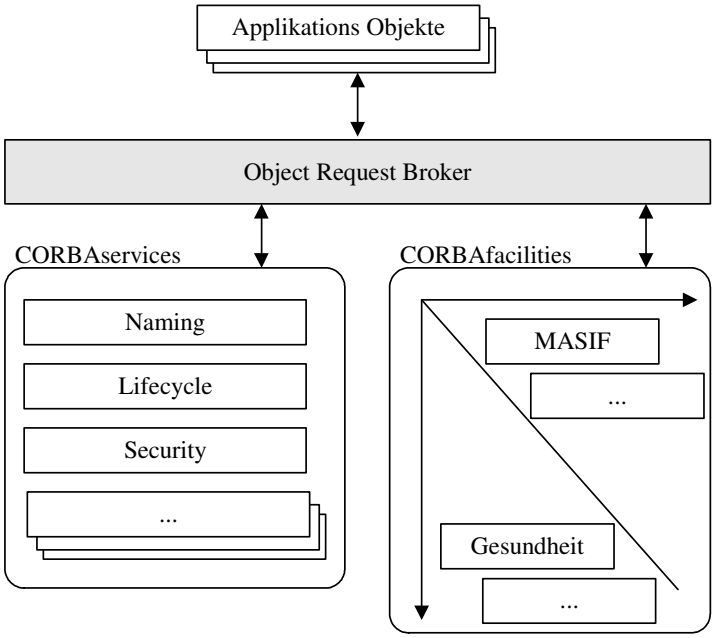


Abbildung 3.4: CORBA und MASIF (angelehnt an [PPW02])

Abbildung 3.4 zeigt die Struktur eines CORBA Systems. Zentrales Element ist der *Object Request Broker* (ORB) der eine betriebssystem- und programmiersprachenunabhängige Kommunikation innerhalb eines verteilten Systems ermöglicht. Dem Entwickler stehen eine Reihe von Diensten, *CORBAservices*, zur Verfügung. Anbei einige relevante Beispiele:

- *Naming Service*
  Dieser Dienst ermöglicht es Objekte über einen definierten Namen anzusprechen, er ist eine Art „Telefonbuch“ für CORBA-Objekte.
- *Lifecycle Service*
  Dieser Dienst stellt Operationen zum Kopieren (Migrieren), Verschieben und Löschen von Objekten zur Verfügung.

[4] OMA teilt Objekte in vier Kategorien: CORBAservices, CORBAfacilities, CORBAdomain Objekte und Applikations Objekte, alle Objekte sind über den *Object Request Broker* miteinander verbunden [OMG-INTRO].

- *Security Service*
  Dieser Dienst unterstützt bei der Umsetzung von Sicherheitsanforderungen wie Authentizität, Vertraulichkeit und Zugriffskontrollen.

Neben diesen allgemeinen Diensten gibt es in CORBA noch Dienste für bestimmte Anwendungsbereiche. Diese werden *CORBAfacilities* genannt. Die *CORBAfacilities* werden in domänenspezifische und horizontale Dienste unterteilt. Domänenspezifische Dienste sind Dienste einer bestimmten Branche, z. B. des Gesundheitswesens, horizontale Dienste sind branchenübergreifende Dienste, wie z. B. die *PrintFacility*. MASIF ist ein horizontaler Dienst, der selbst wieder auf den CORBAservices aufbaut.

Der Programmierer definiert mit Hilfe einer *Interface Definition Language* (IDL) die formalen Schnittstellen der Applikation, die dann mit Hilfe eines IDL-Compilers, der vom Hersteller des jeweiligen ORB zur Verfügung gestellt wird, in ein Objektmodell der verwendeten Programmiersprache umgesetzt wird. [CORBA-WIKI], [PPW02]

Aufbau des MASIF-Modells

Das MASIF-Modell geht davon aus, dass ein Agent immer auch mobil sein kann. Es sind zwei Schnittstellen definiert: *MAFAgentSystem* und *MAFFinder*. Beides sind IDL-Schnittstellen. Das *MAFAgentSystem* unterstützt die Administrierung der Agenten und liefert eine standardisierte Schnittstelle für die Migration. *MAFFinder* dient der standardisierten Namensgebung von Agenten und Agentensystemen und dem Auffinden von Agenten. [MBB99]

Ziel von MASIF ist die Interoperabilität zwischen den Agentenplattformen. Abbildung 3.5 zeigt den Aufbau einer MASIF-konformen Agentenplattform. Die Autorität, *Agent Authority*, identifiziert die Person oder Organisation, in dessen Auftrag der Agent handelt, d. h. den Eigentümer. Das Agentensystem, *Agent System*, bezeichnet eine Plattform, die Agenten erzeugen, ausführen, migrieren und terminieren kann. Ein *Agent System Type* definiert das Profil eines Agenten. Handelt es sich z. B. um einen Aglet Agenten, bedeutet das, dass der Agent in Java entwickelt wurde und auch die Java Objektserialisierung verwendet. MASIF unterstützt unterschiedliche Programmiersprachen und Serialisierungsmethoden! Ein Platz, *Place*, ist ein Bereich innerhalb des Agentensystems der z. B. Funktionen zur Zugriffskontrolle zur Verfügung stellen kann [OMG00]. Innerhalb eines Platzes können ein oder mehrere Agenten ausgeführt werden, Agenten können auch von einem Platz zu einem anderen migrieren. Eine Region fasst alle Agentensysteme einer Autorität zusammen und kann als Sicherheitsdomäne angesehen werden. Dadurch kann mehr als eine Plattform ein und dieselbe Person bzw. Organisation vertreten. Agenten, die zur selben Autorität wie die Region gehören, können so z. B. mit höheren Rechten ausgeführt werden. Der Agentenfinder, *Agent Finder,* ist ein Verzeichnisdienst für Agenten, ähnlich den Gelben Seiten, der pro Region definiert ist. [PPW02]

In [OMG00] findet sich ein kurzer Abschnitt mit dem Titel „1.4.6 *Ensuring a Secure Environment for Agent Operations*“, das die bekannten Gefahren eines Agentensystems aufzeigt, mögliche Gegenmaßnahmen vorschlägt und Anforderungen an ein sicheres Agentensystem definiert. Im Abschnitt 2.4 wird das CORBA *Security Service* vorgestellt, das einige, aber nicht alle, der zuvor definierten Anforderungen erfüllt:

„*Although CORBA security does not currently meet all the needs of mobile agent technology, the MAF implementation must use available CORBA security to satisfy its security needs. Future versions of CORBA security should address these issues*.“ [OMG00, Abschnitt 2.4]

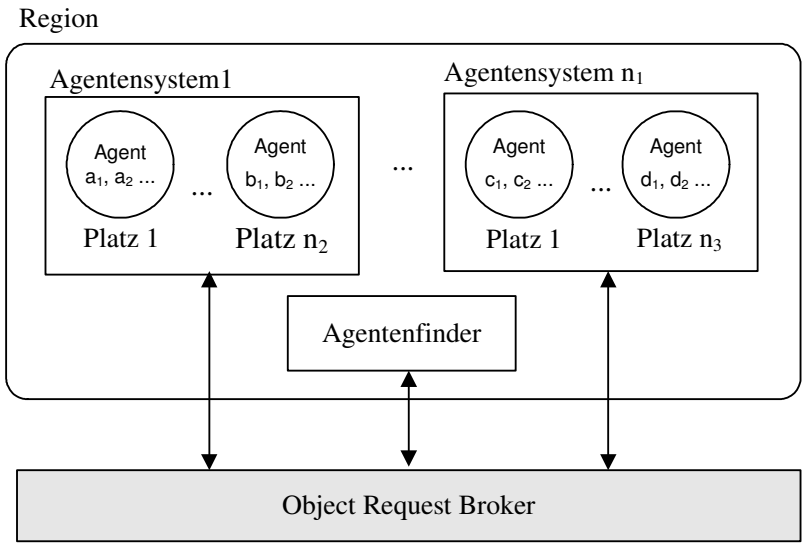


Abbildung 3.5: MASIF-konforme Plattform (angelehnt an [PPW02])

Zurzeit unterstützt MASIF aus sicherheitstechnischer Sicht nur Agenten, die von ihrer Heimatplattform aus nur einmal migrieren (*One-Hop*), d. h. ein Wechsel von dieser ersten Wirtsplattform zu einer weiteren Wirtsplattform (*Multi-Hop*) wird nicht unterstützt:

„*When an agent takes a multi-hop travel which travels between more than two security domains ... the security issues become complex. Most security systems today deal only with security between two domains, which is single-hop travel. The mobile agent community should delay standardizing multi-hop security of mobile agents until security systems can handle the problem.*“ [OMG00, Abschnitt 1.1.2]

3.7.4 FIPA und MASIF im Vergleich

FIPA und MASIF verfolgen unterschiedliche Ziele, daher ist ein direkter Vergleich schwierig. FIPA hat die Standardisierung der gesamten Infrastruktur rund um Agentensysteme zum Ziel, Mobilität ist hier nur ein Randbereich. MASIF dagegen ist ein Standard für mobile Agenten, hier wird nicht die gesamte Infrastruktur, sondern nur die Interoperabilität zwischen den Agentenplattformen betrachtet. Und doch finden sich Gemeinsamkeiten:

- Das FIPA *Agent Management System* ist vergleichbar mit dem MASIF *Agent System*.
  Beide bieten Schnittstellen für grundlegende Funktionen wie Erzeugen, Löschen und Migrieren von Agenten an.

- Der FIPA *Directory Facilitator* ist vergleichbar mit dem MASIF *MAFFinder*.
  Beide definieren Schnittstellen für das Auffinden von Agenten auf Grund bestimmter Eigenschaften.
- Der FIPA *Agent Communication Channel* ist vergleichbar mit dem *Object Request Broker*.

Auf Grund dieser Gemeinsamkeiten schlossen FIPA und OMG 1999 ein Abkommen Standards im Bereich Agententechnologie koordiniert weiterzuentwickeln. Ein entsprechendes *Liaison Proposal OMG-FIPA* findet sich unter http://www.objs.com/agent/omg-fipa-liaison-4.html (abgerufen im September 2008). [PPW02]

4. Sicherheitsanalyse von Agentensystemen

"Friendship is constant in all other things
Save in the office and affairs of love:
Therefore all hearts in love use their own tongues;
Let every eye negotiate for itself
And trust no agent."
William Shakespeare, Much Ado About Nothing

Dieses Kapitel analysiert die Gefahren und möglichen Angriffe, denen ein mobiles Agentensystem ausgesetzt ist. Die Unterteilung erfolgt dabei nach Angreifer und Ziel. Die Angreifer können böswillige Plattformen, böswillige Agenten oder böswillige Systemfremde von außen, d. h. von außerhalb des mobilen Agentensystems, sein. Ihre Angriffe richten sich gegen Plattformen oder mobile Agenten, gelegentlich sogar gegen beide gleichzeitig. Obwohl die jeweils aufgezählten Angriffsarten keinen Anspruch auf Vollständigkeit stellen können - man ist nie davor gefeit, dass ein Angreifer eine noch unbekannte Systemlücke entdeckt - liefern sie einen Überblick der häufigsten und wichtigsten Attacken. Für die Entwicklung eines „sicheren" mobilen Agentensystems ist es unbedingt notwendig zu verstehen, welcher Art von Angriffen das System ausgesetzt sein kann. Sind die Gefahren bekannt, können Entwickler und Agenteneigentümer entsprechende Gegenmaßnahmen setzten.

4.1 Angriffe böswilliger Plattformen

Mobile Agenten benötigen zur Erfüllung ihrer Aufgaben die Unterstützung ihrer Wirtsplattformen. Böswillige Plattformen können Agenten diese Unterstützung verweigern. Sie können auch versuchen aus dem Programmcode oder den Daten des Agenten für sie relevante Informationen auszulesen oder gar Teile des Agenten zu manipulieren. Manipulierte Agenten können auch dazu verwendet werden, dass Plattformen sich gegenseitig attackieren.

4.1.1 Plattform gegen Agent

Greift eine Plattform einen Agenten an, kann sie damit mehrere Ziele verfolgen. Sie kann dies tun, um sich selbst einen Vorteil zu verschaffen oder um anderen Schaden zuzufügen. Zu ihrem eigenen Vorteil kann die Plattform versuchen an für sie wertvolle Informationen zu gelangen, wie z. B. die Entscheidungsalgorithmen oder Kaufkriterien des Agenten. Sie kann diese gewonnen Informationen

nutzen, um auf Anfragen des Agenten entsprechend zu reagieren, etwa mit einem Angebot, das dieser nicht ablehnen kann, oder aber sie nutzt dieses Wissen um den Agenten für sie vorteilhaft zu manipulieren, indem sie z. B. die Kaufkriterien ihren Vorstellungen anpasst. Sie kann den Agent aber auch so manipulieren, dass er seinem Eigentümer oder anderen Wirtsplattformen Schaden zufügt, indem er z. B. anstatt der gewünschten zwei Kartenreservierungen nun hundert Stück reserviert.

Angriffe gegen Agenten können sich gegen ihren Programmcode und/oder ihre Daten richten:

Der Programmcode enthält Informationen, die der Agent zur Bewältigung seiner Aufgabe benötigt. Darin sind z. B. Such- oder Entscheidungskriterien abgebildet, die für eine böswillige Plattform von Interesse sein können. Für gewisse Teile des Programmcodes gilt daher die Anforderung an Vertraulichkeit. Die Hauptanforderung an den Programmcode ist allerdings Integrität, so dass eine Manipulation des Agenten rechtzeitig auffallen und dadurch Schaden verhindert werden kann.

Die Daten eines Agenten können in statische und dynamische Daten unterteilt werden. Die statischen Daten bleiben während der gesamten Laufzeit des Agenten unverändert. Man unterscheidet zwischen öffentlichen und privaten statischen Daten. Öffentliche statische Daten sind für Wirtsplattformen lesbar, private statische Daten sind nur für den Agenten selbst lesbar. Die Hauptanforderung an die öffentlichen statischen Daten ist Integrität, die Hauptanforderung an die privaten statischen Daten ist Integrität und Vertraulichkeit. Ein Beispiel für öffentliche statische Daten ist z. B. der Name des Fluges, für den der Agent das günstigste Angebot suchen soll, Beispiele für private statische Daten sind etwa die Reiseroute des Agenten oder die Kriterien nach denen er die endgültige Kaufentscheidung trifft. [BR05]

Dynamische Daten sind Daten, die der Agent im Laufe seines Lebens sammelt. Sie lassen sich ebenfalls in öffentliche und private dynamische Daten unterteilen. Wie bei den statischen Daten sind auch die öffentlichen dynamischen Daten für alle Plattformen lesbar (Integrität), die privaten dynamischen Daten jedoch lediglich für den Agenten selbst (Integrität und Vertraulichkeit). So gehören z. B. die Liste der bereits besuchten Plattformen und die dabei ausgehandelten Preise zu den privaten dynamischen Daten. [BR05]

Nachfolgend eine Übersicht der möglichen Angriffe „Plattform gegen Agent“ nach [BR05] und [JK99].

Eavesdropping

Allgemein versteht man unter *Eavesdropping* das Ausspionieren von Informationen aller Art durch Belauschen einer Kommunikation. Böswilligen Plattformen bieten sich hier aber noch viel mehr Möglichkeiten. Sie können nicht nur die Kommunikation des Agenten belauschen, sie können auch jede Instruktion, die der Agent ausführt, beobachten. Sie können diese Beobachtungen interpretieren und in Zusammenhang mit den vom Agenten auf der Plattform generierten Daten bringen. Auf diese Weise können sie z. B. Verhandlungsstrategien des Agenten ausspionieren. *Eavesdropping* ist somit nicht nur ein Angriff auf Daten sondern auch auf den Programmcode. Da bei einem *Eavesdropping*-Angriff keine Daten manipuliert werden, handelt es sich hierbei um eine passive Attacke. Ein *Alteration*-Angriff geht einen Schritt weiter.

Alteration

Ein *Alteration*-Angriff ist eine aktive Attacke, es kommt zu einer Manipulation von Daten. Eine böswillige Plattform kann z. B. Teile der dynamischen Daten, d. h. Teile der Daten, die der Agent während seiner Lebenszeit sammelt, verändern. So kann sie z. B. die Preisangebote der bisher besuchten Plattformen so manipulieren, dass ihr eigenes Angebot dadurch zum attraktivsten wird. Sie kann aber auch versuchen, die statischen Daten des Agenten zu ändern, z. B. indem sie die Reiseroute des Agenten so manipuliert, dass dieser bestimmte Konkurrenzangebote nicht mehr einholen kann. Verändert eine böswillige Plattform den Programmcode eines Agenten, ist dies ebenfalls ein *Alteration*-Angriff.

Verwendet das mobile Agentensystem einen verteilten Namensdienst (siehe Abschnitt 3.6) muss der für den Agenten zuständige Namensdienst aus dem Namen des Agenten hervorgehen. Manipuliert ein Angreifer den Namen des Agenten, kann er dabei nicht nur die ID des Agenten verändern, sondern auch den für ihn zuständige Namensdienst. Dies führt dazu, dass der Agent keine seiner eigentlichen Nachrichten mehr erhält und kann sogar soweit führen, dass der Agent nicht mehr ausgeführt wird (*Denial Of Service*). [R01c]

Eavesdropping- und *Alteration*-Angriffe können durch die Anwendung von *Time-Limited Black Boxes* und *Encrypted Functions* verhindert werden. Beide Verfahren, die im Abschnitt 6.3 näher vorgestellt werden, verhindern, dass ein Angreifer die im mobilen Agenten mitgeführten Daten erfolgreich auslesen kann. *Time-Limited Black Boxes* verschleiern den Sourcecode derart, dass eine böswillige Plattform durch Beobachtung der einzelnen Instruktionen des Agenten keine Rückschlüsse mehr auf seine mitgeführten Daten oder Algorithmen ziehen kann. Wendet ein Agent *Encrypted Functions* an, finden alle seine Berechnungen auf der Wirtsplattform in verschlüsselter Form statt, nur die Heimatplattform kennt den dazugehörigen Schlüssel und ist in der Lage, die dabei entstandenen Ergebnisse zu entschlüsseln.

Um ein Verändern der ID des Agenten zu verhindern, muss diese ID vom Eigentümer des Agenten bestätigt und von den Plattformen überprüfbar sein, z. B. durch Verwendung einer digitalen Signatur. [R01c]

Masquerading

Eine böswillige Plattform kann sich gegenüber dem Agenten falsch identifizieren. Sie „versteckt“ sich hinter einer falschen Identität, daher wird dieser Angriff auch *Masquerading*-Angriff genannt. Auf die Anfrage des Agenten antwortet die Plattform mit einer falschen ID z. B. mit dem Namen einer Konkurrenzplattform, um so an deren geheime Daten zu gelangen. Sie kann sich aber auch als Heimatplattform des Agenten ausgeben und den Agenten dahingehend täuschen, dass er ihr alle Ergebnisdaten seines Auftrags mitteilt. Eine *Masquerading*-Attacke verursacht einen doppelten Schaden. Der angegriffene Agent wird geschädigt und die vertrauenswürdige Plattform, deren ID unerlaubterweise verwendet wurde, verliert an Reputation.

Masquerading kann durch *Secure Itinerary Recording* aufgedeckt werden (siehe Abschnitt 6.3.6). Dabei überwachen sich zwei Agenten gegenseitig und überprüfen, ob ihre aktuelle Wirtsplattform auch wirklich die ist, die von ihrer vorhergehenden Wirtsplattform als Zielplattform angegeben wur-

de. Stellt ein Agent fest, dass diese Informationen nicht übereinstimmen, meldet er der Heimatplattform, dass es zu einem Angriff kam.

Denial Of Service

Die böswillige Plattform kann dem Agenten den Zugang zu wichtigen Ressourcen verweigern und ihn so daran hindern seinen Auftrag auszuführen. Die Plattform verweigert dem Agenten ihre Dienste, daher der Name *Denial Of Service*. Sie kann z. B. seine Anfragen ignorieren oder sich sogar ganz weigern den Agenten auszuführen oder zu migrieren. Sie kann aber auch seine Aufforderung zur Migration solange ignorieren, bis eine ihm vorgegebene Zeitspanne abläuft und der Agent keine weiteren Angebote mehr einholt und das bisher günstigste Angebot, das der böswilligen Plattform, akzeptiert.

Mit Hilfe von *Secure Itinerary Recording* (siehe Abschnitt 6.3.6) lassen sich auch *Denial-Of-Service*-Angriffe der Plattform aufdecken. Sobald einer der beiden kooperierenden Agenten feststellt, dass sich sein Partneragent innerhalb einer gewissen Zeitspanne nicht mehr bei ihm gemeldet hat, informiert dieser die Heimatplattform über einen möglichen *Denial-Of-Service*-Angriff.

Cut And Paste

Mit Hilfe eines *Cut-And-Paste*-Angriffs kann eine böswillige Plattform geheime Daten entschlüsseln, obwohl sie nicht in Besitz des dazugehörigen Schlüssels ist. So kann sie z. B. die öffentlichen statischen Daten eines Agenten ausspionieren, obwohl diese nur für eine konkurrierende Wirtsplattform lesbar sind. Ein Beispiel: Ein Agent führt statische Daten mit sich, die nur Plattform A lesen soll, d. h. diese Daten sind mit dem öffentlichen Schlüssel der Plattform A verschlüsselt. Die böswillige Plattform B kopiert die verschlüsselten Daten und fügt sie in ihren eigenen Agenten ein, den sie zu A migriert. Der Agent lässt diese Daten nun von A entschlüsseln und bringt die nun in Klartext vorhandenen Daten zurück zu Plattform B. Auf diese Art kann B zu geheimen Informationen gelangen, die nur für A bestimmt sind. B ist dabei weder im Besitz des öffentlichen, noch des privaten Schlüssels von A.

Cut-And-Paste-Angriffe der Plattform gegen einen Agenten können durch Anwendung der *Target Agencies* Methode verhindert werden (siehe Abschnitt 6.3.8). Der Agent authentifiziert sich gegenüber der Plattform über eine von seinem Eigentümer ausgestellte digitale Signatur. Diese Signatur wurde über einen statischen Kern des Agenten berechnet, der diesen Agenten eindeutig identifiziert.

4.1.2 Plattform gegen Plattform

Plattformen können sich gegenseitig attackieren. Zu diesem Zweck bedienen sie sich manipulierter Agenten.

Alteration

So kann eine Plattform z. B. den Programmcode oder die Daten eines zu migrierenden Agenten manipulieren oder gar löschen. Diese Manipulation schadet nicht nur dem Agenten, sondern kann auch

der nachfolgenden Plattform Schaden zufügen, wie ein Beispiel aus [FGS96] zeigt. Hier wird der zu migrierende Agent derartig verändert, dass er auf der folgenden Wirtsplattform, statt der eigentlich gewünschten zwei Sitzplätze im Flugzeug, gleich hundert Sitzplätze reserviert und so die Wirtsplattform daran hindert weitere Verträge abzuschließen, da sie ihre Kapazität an vorhandenen Sitzplätzen dadurch bereits erschöpft hat. [BR05]

Eine *Alteration* kann durch Anwendung einer *State-Appraisal*-Funktion verhindert werden. Eine genaue Erklärung folgt in Abschnitt 6.2.7.

Masquerading

Eine böswillige Plattform kann sich aber auch als jemand ausgeben, der sie nicht ist. Sie kann z. B. durch Vorspiegeln einer falschen Identität erreichen, dass sie von der Zielplattform des Agenten als vertrauenswürdige Plattform identifiziert wird und somit die Migration des von ihr kommenden Agenten erlaubt wird. Dieser Agent könnte wiederum manipuliert sein und die Zielplattform, ähnlich wie im vorangegangenen Beispiel, zu schädigen versuchen. [BR05]

Das *Masquerading* einer Plattform kann durch die Anwendung einer digitalen Signatur oder durch eine asymmetrisch oder hybrid verschlüsselte Kommunikation verhindert werden. Der Empfänger der Nachricht muss sich darauf verlassen können, dass der Sender der Nachricht der alleinige Besitzer des dazugehörigen privaten Schlüssels ist. Die Verwendung einer *Public Key Infrastructur*, PKI (siehe Abschnitt 3.1.4), dient als zusätzliche Absicherung und verhindert, dass systemfremde Schlüssel zur Anwendung gelangen.

4.2 Angriffe böswilliger Agenten

Böswillige Agenten müssen nicht immer aktiv manipuliert worden sein. Eine fehlerhafte Programmierung kann auf einer Wirtsplattform ebenfalls Schaden verursachen, z. B. durch übermäßigen Ressourcenverbrauch. Für die Plattform stellt es aber keinen Unterschied dar, ob der Schaden auf Grund eines Programmierfehlers oder auf Grund eines geplanten Angriffs eintrat. Aus ihrer Sicht sind beides Attacken auf ihre Sicherheit.

4.2.1 Agent gegen Plattform

Agenten können sich gegenüber ihrer Wirtsplattform böswillig verhalten. Dies kann, wie bereits erwähnt, auf Grund eines Programmfehlers geschehen, oder weil sie aktiv manipuliert wurden. Diese Manipulation kann von einem „menschlichen“ Angreifer oder auch von einer anderen Plattform durchgeführt worden sein.

Angriffe gegen Plattformen können sich gegen die Plattform selbst oder gegen die auf ihr befindlichen Agenten richten. Wird ein Agent auf einer Plattform angegriffen, verliert diese dadurch ihren guten Ruf und ist somit ebenfalls geschädigt. Angriffsziele sind neben den Agenten auch die Informationen, die sich auf der Plattform befinden, wie z. B. Datenbanken. Angreifer können auch die Ressourcen der Plattform übermäßig in Anspruch nehmen, wie z. B. Speicherplatz, CPU-Zeit oder

Netzwerkbandbreite. Es kann aber auch versucht werden, die Plattform in ihrer Arbeit zu behindern oder diese vollständig zu beenden, indem die Plattform terminiert wird.

Nachfolgend eine Übersicht der möglichen Angriffe „Agent gegen Plattform“ nach [NPR06], [J99] und [JK99]:

Masquerading

Ein böswilliger Agent kann sich vor seiner Wirtsplattform tarnen und sich ihr gegenüber z. B. mit dem Agentennamen oder dem Eigentümernamen eines vertrauenswürdigen Agenten authentifizieren und so an geheime Informationen gelangen, die nicht für ihn bestimmt sind. Indem er sich bei der Wirtsplattform mit einer falschen ID anmeldet erhält er auch Zugriffsrechte, die mit dieser ID verbunden sind. Auf diese Weise kann er Zugang zu sensiblen Informationen und Ressourcen erhalten. In einem solchen Fall gibt es mehrere Geschädigte, einerseits die attackierte Plattform, andererseits den vertrauenswürdigen Agenten, der durch die böswillige Anwendung seiner ID seinen guten Ruf verliert.

Masquerading kann durch Verwendung eines eindeutigen Agentennamens, der von den statischen Daten des Agenten abhängig ist und von der Plattform selbst überprüft werden kann, verhindert werden (siehe Abschnitt 3.6).

Unauthorized Access

Indem sich der Agent unter einem falschen Namen tarnt, kann er, wie bereits erwähnt, Zugriffsrechte nutzen, die ihm die Plattform normalerweise nicht gewähren würden. Ein böswilliger Agent kann aber auch versuchen Lücken im System zu finden, um auf diese Weise nicht autorisierten Zugang zu Ressourcen und Informationen der Plattform zu erhalten, d. h. einen *Unauthorized-Access*-Angriff durchführen. Das gleichzeitige „Tarnen“ unter fremden Namen (*Masquerading*) hätte hier den Zweck, die Schuld von sich zu weisen. Je nachdem wieweit einem böswilligen Agenten der Zugang zum System gelingt, kann es sogar geschehen, dass der Agent die gesamte Plattform terminiert und dadurch auch andere Agenten schädigt.

Eine Plattform kann sich vor *Unauthorized-Access*-Angriffen z. B. durch Verwendung einer Sandbox und durch strikte Zugriffskontrollen (*Access Control*) schützen (siehe Abschnitt 6.2.1 und 6.2.3). Eine Sandbox ist ein eigens abgegrenzter Bereich mit eigenen Sicherheitsrichtlinien, die Ausführung des Agenten findet dabei nur innerhalb dieser Sandbox statt.

Denial Of Service

Wie bereits erwähnt, können Sicherheitslücken der Plattform soweit gehen, dass ein Agent diese sogar terminieren kann. Böswillige Agenten können aber auch versuchen einzelne Plattformdienste außer Kraft zu setzen bzw. so stark zu überlasten, dass diese ihre Aufgaben nicht mehr korrekt erfüllen können. Häufig geschieht dies auch auf Grund einer fehlerhaften Programmierung des Agenten. Derartige *Denial-Of-Service*-Angriffe schädigen nicht nur die Plattform, sondern auch alle auf ihr laufenden Agenten. Wird z. B. der Kommunikationsdienst überlastet, werden dadurch auch andere Agenten daran gehindert ihre Heimatplattformen zu kontaktieren.

Bestimmte Arten von *Denial-Of-Service*-Angriffen eines Agenten gegen seine Wirtsplattform können mit Hilfe von *Resource Reification* (siehe Abschnitt 6.2.9) verhindert werden. Die Plattform überwacht dabei ihre Ressourcen und kann gegebenenfalls regelnd eingreifen.

4.2.2 Agent gegen Agent

Ein böswilliger Agent kann die Sicherheitslücken anderer Agenten ausnutzen oder gezielte Attacken gegen sie durchführen. Besonders zu beachten ist, dass einige Agentenplattformen so entworfen wurden, dass bestimmte Plattformkomponenten selbst Agenten sind. Diese Agenten ermöglichen z. B. Festplattenzugriffe oder plattformübergreifende Kommunikation. Zu berücksichtigen ist auch, dass manche Agentenplattformen eine direkte Kommunikation zwischen den Agenten erlauben, wogegen andere Implementierungen wiederum fordern, dass die Kommunikation nur über die Plattform oder einen eigenen Kommunikationsagenten durchgeführt wird. Abhängig von der Plattformarchitektur kann es sich somit entweder um einen Angriff Agent gegen Agent oder Agent gegen Plattform handeln, hier lassen sich die Grenzen nicht immer klar ziehen.

Nachfolgend eine Übersicht der möglichen Angriffe „Agent gegen Agent“ nach [BR05] und [JK99]:

Eavesdropping

Gelingt es einem böswilligen Agenten, die Kommunikation zwischen zwei weiteren Agenten abzuhören, so kann er dadurch in Besitz sensibler Daten gelangen. Der böswillige Agent kann aber auch den Verkehrsfluss zwischen den beiden Agenten analysieren (*Traffic Analysis*) und versuchen aus dem Kommunikationsmuster Rückschlüsse zu ziehen.

Eavesdropping kann z. B. durch Anwendung einer hybriden Verschlüsselung verhindert werden.

Masquerading

Ein böswilliger Agent kann sich hinter einer falschen Identität verstecken und so den angegriffenen Agenten zur Herausgabe geheimer Daten bewegen, z. B. private Informationen oder Kontonummern. Wie bereits mehrfach erwähnt, gibt es bei einem solchen Angriff zwei Geschädigte, den attackierten Agenten und den vertrauenswürdigen Agenten, dessen Identität für den Angriff missbraucht wurde. Die falsche Identität kann einem böswilligen Agenten aber auch zu billigen Ressourcen verhelfen, etwa wenn die Plattform bestimmte Dienste in Rechnung stellt.

Um ein *Masquerading* zu verhindern, müssen sich die kommunizierenden Agenten eindeutig gegeneinander authentifizieren. Da Agenten aber keine Geheimnisse mit sich führen können, wie z. B. einen eigenen privaten Schlüssel, ist dies schwierig. Plattformen sind in Besitz des Agenten und könnten sich, falls die implizite Methode zur Namensgebung angewandt wurde, dessen eindeutigen Namen aus seinen statischen Daten selbst berechnen. Agenten haben diese Möglichkeit nicht. Hier können zur Authentifizierung die Wirtsplattformen oder die Heimatplattformen mit einbezogen werden.

Unauthorized Access

Diese Angriffe beziehen sich auf Agenten, die auf derselben Plattform verweilen. Gelingt einem böswilligen Agenten der direkte Zugriff auf einen anderen Agenten, kann er dadurch einigen Schaden verursachen. Erhält er Zugriff auf öffentliche Methoden des Agenten, kann er z. B. ein Reset durchführen und so den Agenten wieder in seinen Grundzustand versetzen. Gelingt einem böswilligen Agenten gar der Zugriff auf den Sourcecode, kann er aus einem ehemals vertrauenswürdigen Agenten einen böswilligen Agenten machen. Einige Plattformen stellen ihre Dienste über eigene Agenten zur Verfügung. Wird ein derartiger Agent attackiert, z. B. der Agent, der den Kommunikationsdienst zur Verfügung stellt, ist dies gleichzeitig auch ein Angriff gegen die Plattform.

Unauthorized-Access-Angriffe können durch gegenseitige Authentifizierungen verhindert werden.

Denial Of Service

Ein böswilliger Agent kann einen anderen Agenten durch das andauernde Abschicken von Nachrichten an der Erledigung seiner eigentlichen Aufgabe hindern. Der angegriffene Agent kann diese Nachrichten zwar blocken, aber auch der Vorgang des Blockens erfordert Prozessorzeit. Auf Plattformen, die sich ihre CPU-Zeit bezahlen lassen, kann sich ein derartiger Angriff für den Agenten, bzw. dessen Eigentümer, somit auch finanziell negativ auswirken. Ein böswilliger Agent kann aber auch unsinnige Nachrichten verschicken und so den angegriffenen Agenten von der Erledigung seiner Aufgabe abhalten. Der Angreifer könnte aber auch einen eigenen Agenten unter fremdem Namen in das System einbringen und so die an diesen Agenten gerichtete Kommunikation abfangen. Da der eigentliche Agent dadurch keine Nachrichten mehr erhält, kann dieser seine Aufgabe nicht mehr erfüllen. Er kann zwar noch Nachrichten an seine Heimatplattform absenden, deren Antwort aber nicht mehr empfangen. [R01c]

Denial-Of-Service-Angriffe können, wie bereits erwähnt, geblockt werden. Um zu Verhindern, dass der Name eines Agenten „gekidnappt“ und bei einem böswilligen Agenten eingesetzt wird, kann der Name des Agenten implizit vergeben werden, d. h. der Agentenname wird mit Hilfe einer Hash-Funktion aus dem statischen Kern, z. B. dem Programmcode, des Agenten berechnet und vom Agenteneigentümer signiert (siehe Abschnitt 3.6). Diese Verbindung zwischen Name und Signatur bewirkt eine Eindeutigkeit des Namens.

Repudiation

Von einem *Repudiation*-Angriff spricht man, wenn einer der beteiligten Agenten leugnet, dass eine bestimmte Kommunikation oder Transaktion je stattgefunden hat. Dies kann böswillig geschehen, oder aber auf Grund eines technischen Problems, etwa weil es zu einer Unterbrechung der Kommunikation kam. Die Wirtsplattform der Agenten kann nicht verhindern, dass es zu solchen Vorfällen kommt, sie kann aber dazu beitragen, dass im Falle der Uneinigkeit zwischen zwei Agenten entsprechende Logdateien zur Verfügung stehen, die bei der Beseitigung des Streitfalles helfen können.

Leugnet ein Agent eine stattgefundene Transaktion, muss bewiesen werden können, dass diese Transaktion sehr wohl stattfand. Um dies zu beweisen müssen einerseits die IDs des beteiligten Agenten

sichergestellt sein, d. h. die beiden Agenten müssen sich gegenseitig authentifiziert haben, andererseits muss die Transaktion, möglichst signiert, in einer Logging-Tabelle aufgezeichnet sein.

4.3 Angriffe böswilliger Systemfremder

Die in einem Agentensystem stattfindende Kommunikation kann auch von Systemfremden belauscht werden (*Eavesdropping*). Die Angreifer können versuchen an Hand der Kommunikationsmuster diverse Rückschlüsse zu ziehen (*Traffic Analysis*). Diese Rückschlüsse können Informationen liefern, die z. B. für *Cut-And-Paste*-Angriffe verwendet werden können. Mit Hilfe einer *Cut-And-Paste*-Attacke könnte ein Angreifer versuchen an geheime Daten, wie z. B. Kontonummern, zu gelangen.

Die im Abschnitt 3.6 vorgestellten Namensdienste, die eine transparente Kommunikation mit den mobilen Agenten ermöglichen, gelangen durch ihre Dienste an viel Wissen über den Agenten. Sie können die Routen, der von ihnen verwalteten Agenten nachverfolgen und mit Hilfe dieser Information u. U. auch vorausberechnen. Sie wissen welcher Eigentümer welchen Agenten besitzt und kennen die Plattformen, die diese besuchen. Die Privatsphäre der Eigentümer der Agenten wird dadurch aufgeweicht. [R01c]

Eavesdropping kann durch die Anwendung von Verschlüsselungsalgorithmen verhindert werden. *Traffic Analysis* wird durch die Verwendung von Nachrichten gleichbleibender Länge oder durch die Verwendung von Protokollen, die je Kommunikationssession eine gleichbleibende Anzahl an Anfragen und Antworten verwenden, erschwert. *Cut-And-Paste*-Angriffe können durch Verwendung von digitalen Signaturen in Kombination mit Zeitangaben, Zählern oder Zufallszahlen verhindert werden. Die implizite Namensgebung von Agenten (siehe Abschnitt 3.6) verhindert, dass für Namensdienste der Zusammenhang zwischen Eigentümer und Agent sichtbar wird und so die Privatsphäre des Agenteneigentümers eingeschränkt wird.

4.4 Zusammenfassung

Die zahlreichen Attacken, die gegen ein mobiles Agentensystem möglich sind, zeigen wie wichtig das Thema „Sicherheit“ in diesem Zusammenhang ist.

Tabelle 1 stellt eine Übersicht, der eben vorgestellten Angriffe dar. Die Tabelle stellt jedem Angriff mindestens eine Sicherheitstechnologie gegenüber, die als Gegenmaßnahme herangezogen werden kann. Einige der darin erwähnten Gegenmaßnahmen, wie z. B. die Verwendung digitaler Signaturen, wurden bereits als „State Of The Art“ im Kapitel 3 vorgestellt, andere, wie z. B. *Time-Limited Black Boxes* oder *Secure Itinerary Recording*, sind speziell auf mobile Agentensysteme anzuwendende Methoden und werden im Kapitel 6 noch genauer vorgestellt.

Tabelle 1: Gegenüberstellung von Angriffen und kryptographischen Gegenmaßnahmen

Art des Angriffs		**Sicherheitstechnologie**
Plattform gegen Agent	*Eavesdropping* *Alteration*	*Time-Limited Black Boxes* *Encrypted Functions*
	Masquerading *Denial Of Service*	*Secure Itinerary Recording*
	Cut And Paste	*Target Agencies*
Plattform gegen Plattform	*Alteration*	*State Appraisal*
	Masquerading	Digitale Signatur
Agent gegen Plattform	*Masquerading*	Impliziter Agentenname
	Unauthorized Access	*Sandbox* *Access Control*
	Denial Of Service	*Resource Reification*
Agent gegen Agent	*Eavesdropping*	Hybride Verschlüsselung
	Masquerading *Unauthorized Access*	Authentifizierung
	Denial Of Service	Blocken
	Reputation	Authentifizierung und Logging
Angriff durch Systemfremde	*Eavesdropping*	Verschlüsselungsalgorithmen
	Traffic Analysis	Nachrichten gleicher Länge Regelmäßige Protokolle
	Cut And Paste	Digitale Signatur über Zeitangabe oder Zähler

5. Mobile Agentensysteme im Sicherheitsvergleich

„Es gibt keine Sicherheit,
nur verschiedene Grade der Unsicherheit.“
Anton Pawlowitsch Tschechow

Im Folgenden werden jeweils zwei bedeutendsten mobilen Agentensysteme vorgestellt und deren Sicherheitsmaßnahmen miteinander verglichen. Bei den vorgestellten Systemen handelt es sich um JADE und SeMoA sowie Aglets und Grasshopper. Jede dieser Plattformen war zu Testzwecken aktiven Angriffen ausgesetzt. Die dabei verwendeten Methoden zeigen auf, wie Angreifer vorgehen und wo Entwickler mobiler Agentensysteme besondere Sorgfalt walten lassen sollten.

All diese Systeme sind nicht nur Plattformen für mobile Agenten, sondern stellen auch die dabei benötigten Entwicklungs- und Administrierungstools zur Verfügung.

5.1 JADE und SeMoA

JADE ist vermutlich das aktuell am weitesten verbreitete Agentensystem und SeMoA ein Agentensystem, das speziell mit Blick auf Sicherheit und Mobilität entwickelt wurde. Mitarbeiter des Fraunhofer-Instituts für Graphische Datenverarbeitung in Darmstadt (IGD) haben eine Reihe von Tests durchgeführt und beide Systeme bezüglich ihrer Angriffssicherheit in [BHMW08] miteinander verglichen.

5.1.1 JADE

JADE ist die Abkürzung für *Java Agent Development Framework*. JADE ist eine in Java geschrieben *Open Source* Plattform und wird von Telecom Italia Lab (TILAB) vertrieben. Sie baut auf dem FIPA-Standard auf und hat eine reichhaltige Klassenbibliothek, die u. a. Prototypen für verschiedene Verhaltensobjekte und für diverse FIPA-Interaktionsprotokolle enthält. Die Instanz der JADE-Laufzeitumgebung wird Container genannt, ein Container enthält einen oder mehrere Agenten. [S07]

Zur Umsetzung von Sicherheitsanforderungen gibt es in JADE ein eigenes Zusatzpaket das folgende Dienste zur Verfügung stellt:

Sicherheits- und Authentifizierungsdienst (*JADE Security Service*)

Dieser Dienst ermöglicht es, die JADE Plattform multiuserfähig zu betreiben. Es stützt sich dabei auf JAAS (Java *Authentication and Authorization Service*), einen Java Dienst, der Programmen unterschiedliche Rechte zuordnet, abhängig davon, wer das Programm gerade ausführt. [NPR06]. Der Authentifizierungsdienst besteht aus zwei Teilen. Einem *Callback Handler*, um Benutzername und Passwort abzufragen und einem Loginmodul, um die erhaltenen Informationen zu verifizieren. JADE bietet mehrere Varianten eines *Callback Handlers* an, z. B. über *Command Line* oder Dialog. Ebenso gibt es mehrere Varianten des Loginmoduls, betriebssystemabhängig oder unabhängig (Kerberos).

Genehmigungsdienst (*Permission Service*)

Ist der Genehmigungsdienst aktiviert, wird jedem Agenten und jedem Container ein authentifizierter Benutzer zugeordnet. Mit Hilfe von Zugangsrechten werden einzelne Aktionen der Agenten erlaubt oder verhindert. Diese Zugangsrechte sind in einem *Policy File* definiert, die *Policy File* Syntax von Java/JAAS wurde daher um einige Elemente erweitert. Es gibt zwei Arten von *Policy Files.* Das *Policy File* des Hauptcontainers definiert plattformweite Rechte, das Container *Policy File* legt die Rechte eines bestimmten Containers fest. Container *Policy Files* enthalten auch Regeln für den Zugriff auf die lokalen Ressourcen, wie etwa die JVM, das Filesystem oder das Netzwerk. Alle *Policy Files* können mit Hilfe des Java *Policy Tools* administriert werden.

Signaturdienst

Nachrichten können vom Sender mit seinem privaten Schlüssel signiert werden. Der dazugehörige öffentliche Schlüssel wird an die Nachricht angehängt und gemeinsam mit ihr versendet. Mit Hilfe dieses öffentlichen Schlüssels verifiziert der Empfänger die Signatur. Damit wird sichergestellt, dass die Nachricht während ihrer Übertragung nicht verändert wurde.

Verschlüsselungsdienst

Zur Wahrung der Vertraulichkeit können die Nachrichten verschlüsselt werden; dabei kommt ein hybrides Verfahren zum Einsatz. Jede Nachricht wird mit einem eigens erzeugten symmetrischen Schlüssel verschlüsselt. Der symmetrische Schlüssel selbst wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und an die Nachricht angehängt. Der Empfänger, Besitzer des privaten Schlüssels, entschlüsselt den symmetrischen Schlüssel und verwendet diesen zur Entschlüsselung der eigentlichen Nachricht.

5.1.2 SeMoA

SeMoA ist die Abkürzung von *Secure Mobile Agents* [SEMOA]. SeMoA ist eine in Java geschrieben *Open Source* Plattform speziell für mobile Agenten mit dem Schwerpunkt auf der Umsetzung von Sicherheit. Sie wurde am IGD entwickelt. SeMoA stellt Basisfunktionen für die Migration, die Kommunikation und das Aufspüren (*Tracking*) von Agenten zur Verfügung und ist kompatibel zu

anderen Plattformen, etwa Aglets oder JADE, d. h. Aglets oder JADE Agenten können auf einer SeMoA Plattform ausgeführt werden [SEMOA]. Die Plattform selbst hat eine modular aufgebaute Sicherheitsarchitektur, die es ermöglicht, weitere Sicherheitsfilter anzubauen. Die Gesamtarchitektur besteht aus vier Schichten mit unterschiedlichen Sicherheitsmechanismen, um Angriffe gegen Agenten oder die Plattform selbst abwehren zu können.

Zuunterst liegt die Transport Schicht, in der Protokolle wie TLS oder SSL für die Migration der Agenten zum Einsatz kommen. Die zweite Schicht besteht aus eine Reihe von Sicherheitsfiltern, jeder ein- oder ausgehende Agent muss diese Filter passieren. Hier werden Signaturen überprüft, Agenten entschlüsselt und Rechte, die während der Ausführung des Agenten Gültigkeit haben, vergeben. Diese Rechte basieren auf konfigurierbaren Sicherheitsrichtlinien (*role-based*). In Schicht drei erhält jeder Agent seinen eigenen *Class Loader*. Jede Klasse des Agenten, die geladen werden soll, wird zuerst gegen eine Liste von Hashwerten überprüft. Diese Liste wurde vom Eigentümer des Agenten signiert. Die vierte Schicht ist die Sandbox in der der Agent zur Ausführung kommt, hier erhält jeder Agent seine eigene Threadgruppe und jede seiner Aktionen wird von einem Security Manager überwacht, der für die Einhaltung der in Schicht zwei vergebenen Rechte sorgt, z. B. wenn es um den Zugriff auf Dateien oder Verzeichnisse geht.

In SeMoA signiert der Eigentümer die statischen Daten des Agenten und jede Plattform, die der Agent besucht, signiert diesen nochmals. Jeder Agent hat einen global eindeutigen Namen, der von der Signatur seines Eigentümers abgeleitet wird. Diese Eigenschaften erschweren Angriffe auf die Identität oder die Signaturen der Agenten. [BHMW08], [RJP01]

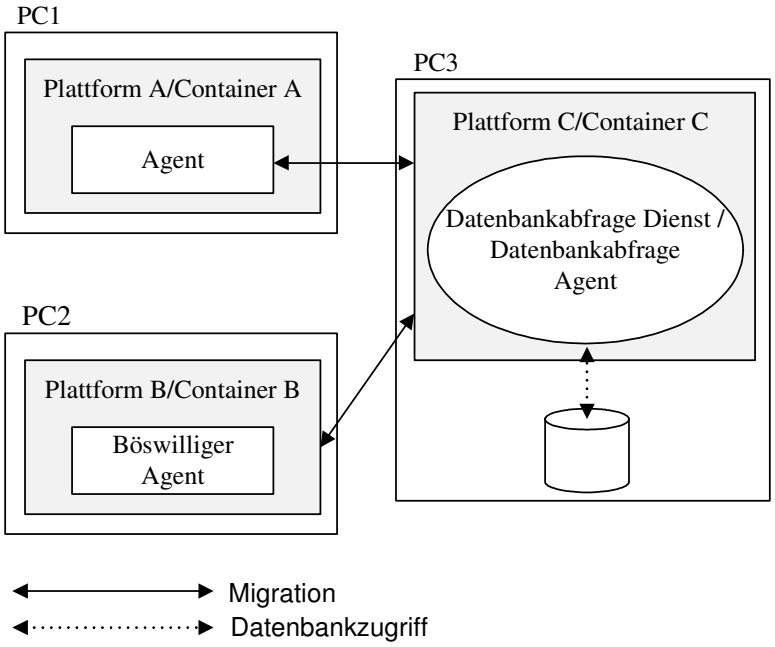


Abbildung 5.1: Architektur des Testsystems [BEHMW06]

5.1.3 Aktiv durchgeführte Angriffe auf JADE und SeMoA

In [BEHMW06] werden die beiden eben vorgestellten Agentensysteme einander gegenübergestellt. Die Mitarbeiter des IGD arbeiteten eine Reihe von Testfällen aus, um die Sicherheit der beiden Plattformen gegenüber Angriffen von böswilligen Agenten zu überprüfen. Böswillige Agenten können die Plattform selbst, aber auch die auf ihr laufenden Agenten attackieren. Bei den Tests wurde JADE Version 3.3 mit dem Sicherheitszusatzpaket (*Security Add-On*) Version 3[1].3 und SeMoA mit der Release vom 12. August 2005 verwendet (semoa complete 050812zip).

Abbildung 5.1 zeigt die verwendete Test Architektur. Es kamen drei PCs unterschiedlicher Hardwaretypen zum Einsatz. PC1 ist Heimatplattform des vertrauenswürdigen Agenten A, PC2 ist Heimatplattform des böswilligen Agenten B. Der vertrauenswürdige Agent A migriert zwecks Datenbankabfrage zur Plattform C und kehrt im Anschluss daran wieder auf seine Heimatplattform zurück. Zur gleichen Zeit migriert auch der böswillige Agent B zur Plattform C und versucht, je nach Testszenario, entweder die Plattform oder den Agenten zu attackieren.

Datenbankzugriffe werden in JADE durch einen eigenen Agenten, in SeMoA aber durch die Plattform selbst, abgewickelt.

Die gewählten Tests deckten drei Arten von Angriffen ab:

- *Denial-Of-Serivce*-Angriffe auf die Plattform
- Angriffe durch nicht autorisierte Zugriffe auf Daten der Plattform
- Angriffe auf andere Agenten

Auf Grund der unterschiedlichen Plattformarchitekturen und Sicherheitskonzepte war es schwierig, Tests zu finden, die auf allen Plattformen die gleiche Aussagekraft haben. Man entschied sich daher gegen eine homogene Testmethode, trotzdem befanden sich auch Tests darunter, die auf beiden Systemen gleich angewandt werden konnten.

Denial-Of-Service-Angriffe auf die Plattform

Mit diesen Tests sollte herausgefunden werden, in wie weit sich die Plattformen durch böswillige Agenten in der Durchführung ihrer eigentlichen Aufgaben behindern lassen, wie etwa durch das Überlasten der CPU oder durch exzessive Verwendung von Betriebssystemressourcen. Dazu wurde der vertrauenswürdige Agent in einer Endlosschleife betrieben und die durchschnittliche Durchführungszeit je Durchlauf gemessen. Unter anderem wurden folgende *Denial-Of-Service*-Angriffe durchgeführt:

- Überlasten der Plattform durch zu viele Agenten
  Dies geschah in JADE durch anhaltendes Klonen des böswilligen Agenten und in SeMoA, da hier kein Klonmechanismus zur Verfügung steht, durch Spammen der Plattformen mit Agenten, d. h. andauerndes Eintreffen neuer Agenten.

- Überlasten der Plattform durch zu viele Serviceanfragen
  Dies geschah in JADE durch Spammen des *Agent-Management-System*-Agenten und in SeMoA durch andauerndes Aufrufen des *Database Access Services*.
- Überlasten der Plattform durch nicht terminierende Agenten
  Dies geschah in JADE durch *Non-Blocking Behaviors* und in SeMoA durch Endlosschleifen.

Angriffe durch nicht autorisierte Zugriffe auf Daten der Plattform

Hier wurde untersucht, ob es einem böswilligen Agenten möglich ist, auf wichtige Funktionen der Plattform zuzugreifen und diese zu sabotieren oder Zugang zu geheimen Daten zu erhalten.

- Abschalten der Plattform
- Modifizieren der Sicherheitsrichtlinien
- Austauschen des JAVA Security Managers
- Entfernen eines Agenten aus dem Namensverzeichnis (*White Pages*) der auf der Plattform verfügbaren Agenten

Angriffe auf andere Agenten

Hier wurde untersucht, ob es einem böswilligen Agenten möglich ist, einen anderen, auf derselben Plattform laufenden Agenten anzugreifen. Diese Testfälle weisen Ähnlichkeiten mit den *Denial-Of-Service*-Testfällen auf.

- Spammen des Agenten mit unsinnigen Anfragen
- „Abschießen“ des Agenten
- Senden einer signierten Nachricht unter einer falschen Absender-ID
  Dies wurde nur in JADE getestet, da dies unter SeMoA nicht möglich ist.
- Verändern des Ergebnisses der Datenbankabfrage
  Dies wurde nur in SeMoA getestet, da dies unter JADE nicht möglich ist.

Ergebnisse für JADE

Für JADE zeigte sich, dass das andauernde Klonen der Agenten und *Non-Blocking Behaviors* nicht verhindert werden können, gleiches gilt für das Spammen eines Agenten mit unsinnigen Anfragen. Des Weiteren deckten die Tests eine erhebliche Sicherheitslücke von JADE auf. Während der Migration des Agenten kann die Eigentümerinformation eines Agenten leicht durch eine falsche Eigentümerinformation ausgetauscht werden. Dadurch wird es dem Agenten möglich, Nachrichten unter falschem Absender zu versenden.

Ergebnisse für SeMoA

Aufgrund der Tatsache, dass schon bei der Entwicklung von SeMoA auf die Umsetzung von Sicherheit hoher Wert gelegt wurde, hielt SeMoA allen Angriffen durch nicht autorisierte Zugriffe und allen Angriffen auf andere Agenten stand. Allerdings zeigte sich, dass bei der Entwicklung eines Agenten auf das „manuelle“ Löschen seines Nachrichtenspeichers vor der Migration geachtet werden sollte, und zwar aus folgendem Grund: Das Spammen eines Agenten ist nur möglich, wenn dieser regelmäßig seine Nachrichten ausliest, anderenfalls erhält der Absender die Information, dass der Empfänger nicht erreichbar ist. So gesehen richtet das Spammen des Agenten nicht viel Schaden an. Da aber alle noch nicht gelesenen Nachrichten den Daten des Agenten hinzugefügt werden, kann es geschehen, dass dieser für eine Migration zu groß wird und somit auf der Plattform „gefangen“ bleibt. Daher sollte ein Agent seine Mailbox vor einer Migration aktiv löschen.

Zusammenfassung

Die durchgeführten Tests zeigten, dass die vorhandenen Sicherheitsmechanismen im Rahmen ihrer Spezifikation wirken. Dies ist aber nur dann sichergestellt, und dieser Punkt ist von äußerster Wichtigkeit, wenn die Sicherheitsmechanismen auch korrekt konfiguriert werden. Kein Agent sollte z. B. Zugang zu Mechanismen haben, die dem Plattform Management dienen. Auf die Administrierung des Systems ist daher besonders zu achten. Restriktive Rechtevergabe kann einige *Denial-Of-Service*-Angriffe verhindern. Leider zeigte sich aber, dass der Großteil der *Denial-Of-Service*-Angriffe kaum vermieden werden kann. Computer mit mehr RAM oder *dual core* Prozessoren schnitten dabei aber, mit Ausnahme eines SeMoA-Testfalls, besser ab.

Details zu Testfällen und Ergebnissen sind unter [BEHMW06] nachzulesen.

5.2 Aglets und Grashopper

Es finden sich zwar auch aktuelle Artikel, wie z. B. [FA07], die die Sicherheitsaspekte mobiler Agentensysteme einander gegenüberstellen, jedoch sind diese hauptsächlich theoretischer Natur, d. h. die Sicherheitskonzepte werden an Hand der Spezifikationen analysiert und miteinander verglichen. Fischmeiser, Vigna und Kemmerer haben in [FVK01] drei Agentensysteme ausgewählt, deren Sicherheitskonzepte analysiert und diese anschließend aktiv attackiert. Die Angriffe richteten sich dabei hauptsächlich auf die Autorisierungssysteme der Plattformen. Die dabei gefundenen Sicherheitslücken und die Realisierung der Attacken bieten interessante Blickwinkel auf die Sicherheit in mobilen Agentensystemen und zeigen, worauf Entwickler von Agentensystemen unter anderem zu achten haben. Bei den von Fischmeister et al. ausgewählten Agentensystemen handelt es sich um Aglets, Jumping Beans und Grashopper. Im Folgenden werden nur die Ergebnisse von Aglets und Grashopper vorgestellt, da Jumping Beans in der Praxis nicht so häufig zur Anwendung kommt.

5.2.1 Aglets

Aglets wurde zu hundert Prozent in Java programmiert, das zeigt sich auch in seinem Namen, der eigentlich für *Agent Applets* steht. In Aglets wird ein mobiler Agent auch als „Aglet“ bezeichnet. Ursprünglich war Aglets ein kommerzielles Produkt, entwickelt im *IBM Tokyo Research Laboratory*, mittlerweile ist es ebenfalls ein Open Source Projekt. [FA07]

Aglets basiert teilweise auf dem OMG MASIF-Standard. Es unterscheidet stark zwischen mobilen und stationären Agenten. Den stationären Agenten, d. h. den Agenten, die ihre Plattform nie verlassen, wird Vertrauen entgegen gebracht, den mobilen Agenten nicht. Mobile und stationäre Agenten besitzen eine eigene Klassenhierachie und leiten sich nicht von derselben übergeordneten Klasse ab. Der Zugang zu Ressourcen muss über stationäre Agenten implementiert werden. Entgegen der MASIF-Spezifikation unterstützt Aglets nicht mehrere Plätze pro Agentensystem. Ein Platz und ein Agentensystem bilden eine abstrakte Einheit, diese Komponente wird Tahiti genannt. Sollen mehrere Plätze unterstützt werden, muss der Administrator mehrere Tahiti-Server starten, d. h. einen Server je Platz. Entgegen der MASIF-Spezifikation gibt es in Aglets keine Region, somit ist jede Plattform einer eigenen Autorität unterstellt. Tahiti bietet auch Dienste für Kommunikation, Administration, Sicherheit und Transport an und enthält außerdem ein *MAFAgentSystem,* eine standardisierte Schnittstelle, die es ermöglicht, Agenten, die nicht auf Aglets basieren, in einer Agletsumgebung auszuführen. (siehe Abschnitt 3.7.3) [FS00]

Fischmeister et al. analysierten Aglets SDK 1.1, die aktuelle Version ist Aglets SDK 2 [AGLETS]. Aglets ist weiterhin in Entwicklung, die Aglets Homepage (http://www.trl.ibm.com/aglets/; abgerufen im September 2008) wird gewartet und enthält Einträge zu aktuellen Entwicklungen. Allerdings stammt die letzte Spezifikation aus dem Jahr 1998 und bezieht sich auf Aglets SDK 1.1, es gibt aber ein Handbuch für Aglets 2.0.2 aus dem Jahr 2004.

Angriffe auf den Programmcode

Agenten können auf den Programmcode der Plattform nicht direkt zugreifen, daher wurde versucht eine Umgehungslösung zu finden. Fischmeister et al. fanden heraus, dass unter Zuhilfenahme der JAVA *Reflection*-Klasse wichtige Informationen über den Progammcode erhalten werden können. Die *Reflection*-Klasse ermöglicht es in einem Javaprogramm Klassen und Interfaces zu verwenden, die zum Zeitpunkt des Kompilierens noch nicht bekannt sind. Die *Reflection*-Klasse wird z. B. beim dynamischen Laden von Klassen benötigt [REFLECT]. Um den Angriff durchzuführen, muss der Agent zuerst eine *Exception* auslösen. Die *Exception* enthält eine Momentaufnahme des aktuellen Ausführungsstacks. Der Stack wurde analysiert und alle Referenzen auf Klassennamen innerhalb des Stacks wurden zur späteren Verwendung gespeichert. Sobald eine Anzahl von Klassen identifiziert war, wurde mit Hilfe der *Reflection*-Klasse versucht, die Konstruktoren, Methoden und Überklassen der jeweiligen Klassen zu erhalten. Auf diese Art konnten Teile des Programmcodes aufgedeckt werden. Im letzten Schritt wurden die gefundenen Klassen noch nach statischen Methoden und Attributen durchsucht. Diese können für böswillige Agenten nützlich sein, da Agenten auf diese Weise Operationen ausführen können, ohne eine Objektreferenz zu benötigen.

Angriffe auf Sicherheitsrichtlinien

Unter Zuhilfenahme des zuvor beschriebenen Angriffs erhielten Fischmeister et al. Zugang zu den Sicherheitsrichtlinien der Plattform. Sie fanden heraus, dass die Datenbank, in der die Sicherheitsrichtlinien gespeichert sind, über eine statische Methode angesprochen wird. Das bedeutet, dass diese, ohne im eigentlichen Besitz der Objektreferenz auf das notwendige Objekt zu sein, angesprochen werden kann. Ein durchgeführter Schreibvorgang auf die Datenbank wurde vom Security Manager nicht erkannt, d. h. auf diese Weise ließen sich Sicherheitsrichtlinien modifizieren oder gar hinzufügen, ohne dass der Angriff dem System auffiel.

Angriffe auf die graphische Benutzerschnittstelle

Aglets erlaubt es Agenten, nur Fenster mit Warnungen auszugeben. Dies soll verhindern, dass böswillige Agenten Fenster öffnen, die administrativen Plattformfenstern ähneln und etwa zur Eingabe des Benutzerpasswortes auffordern. Auf diese Weise könnte der böswillige Agent in Besitz sensibler Informationen gelangen. Fischmeister et al. fanden heraus, dass auf Grund eines Fehlers in der Implementierung, die Einschränkung auf „nur Warnungsfenster Ausgabe möglich“ nicht umgesetzt wurde und die Agenten in der Lage waren, jegliche Art von Fenstern und Dialogen zu öffnen.

5.2.2 Grasshopper

Grasshopper, ebenfalls in Java programmiert, wurde von GMD FOKUS entwickelt und ist die erste Referenzimplementierung zum OMG-Standard MASIF (siehe Abschnitt 3.7.3). Laut einer Pressemitteilung stellte die Firma IKV++, die Grasshopper vertrieb, die Version 2.1 im Jahr 2000 zum kostenlosen Download auf der Internetseite www.grasshopper.de zur Verfügung [IKV]. Diese Seite wurde in der Zwischenzeit von einer branchenfremden Firma übernommen und auf der Homepage der Firma IKV++ (www.ikv.de) finden sich keine weiteren Hinweise. Die im Folgenden beschriebenen Angriffe wurden auf Version 1.2.2.3 durchgeführt.

Angriffe auf vertrauenswürdigen Code

Ähnlich Aglets verwendet Grasshopper vertrauenswürdige Klassen. Diese Klassen überscheiben den Security Manager, verfügen aber über keinerlei Zugriffsüberprüfungen. Diese Eigenschaft ist eine Sicherheitslücke von Grasshopper, so kann z. B. mit Hilfe der vertrauenswürdigen Klasse *javax.swing.JInternalFrame* die Ausführung der Plattform beendet werden.

Angriffe auf die graphische Benutzerschnittstelle

Fischmeister et al. stellten fest, dass die Methode *checkAwtEventQueueAccess* nicht implementiert wurde. Durch Ausnutzen dieser Sicherheitslücke war es ihnen möglich, Zugriff auf die Ereignisse (*Event Queue*) der graphischen Schnittstelle zu erhalten. Über die Ereignis Referenzierungen erhielten sie Zugriff auf externe, nicht zum Agenten gehörende, graphische Komponenten. Durch Abschicken gefälschter Ereignisse konnten sie diese externen Komponenten bedienen. Auf diese Art wurde ein Angriff über die Tastenkombination „Alt-Shift-Q“ durchgeführt. „Alt-Shift-Q“ ist die Tasten-

kombination, die eine Grasshopper Plattform terminiert. Zuvor erscheint allerdings ein Fenster mit der Bitte um Bestätigung. Während der Attacke wurde das Erscheinen dieser Dialogbox ebenfalls überwacht und ein gefälschter Bestätigungsclick abgeschickt. Auf diese Weise wurde das Autorisierungssystem von Grasshopper umgangen und das Agentensystem heruntergefahren.

Angriffe auf Systemmerkmale

Bei ihren Analysen stellten Fischmeister et al. auch fest, dass die Methode *checkPropertyAccess* keine Sicherheitsüberprüfungen eingebaut hat. Dadurch war es ihnen möglich, Zugriff auf grundlegende Merkmale des Systems zu erhalten und diese auch zu manipulieren. Auf diesem Weg konnten sie z. B. den Namen der Plattform verändern.

Angriffe auf Sicherheitsrichtlinien

Beim Versuch, Zugriff auf die Sicherheitsrichtlinien von Grasshopper zu erhalten, wurde festgestellt, dass auf diese, ähnlich wie bei Aglets, über statische Methoden und Variablen zugegriffen wird. Obwohl für den Zugriff auf das *Policy*-Objekt spezielle Rechte benötigt werden, war es möglich ein neues *Policy*-Objekt anzulegen. Die darin enthaltenen manipulierten Sicherheitsrichtlinien zeigten vorerst keine Wirkung, da die ursprünglichen Sicherheitsrichtlinien noch statisch vorhanden waren. Sobald aber der Systemmanager den Konfigurationsdialog der Sicherheitsrichtlinien das nächste Mal öffnete, wurden die manipulierten Sicherheitsrichtlinien automatisch aktiv.

6. Umsetzung der Sicherheitsanforderungen

"We're not a security guard company.
We sell a 'concept' of security."
Michael Kaye, president of Westec, a residential security company

Bei der Umsetzung der in Abschnitt 3.2 vorgestellten Sicherheitsanforderungen müssen mobiler Agent und Plattform getrennt betrachtet werden. Eine Plattform kann im Falle eines Angriffs eine Art *Fail-Safe* Zustand einnehmen. Erkennt sie den Angriff rechtzeitig, gelangt der böswillige Agent erst gar nicht zur Ausführung. Erkennt sie den Angriff zur Laufzeit, kann sie den Agenten beenden und löschen. Für Agenten gibt es diesen sicheren Zustand nicht. Die Abschnitte 6.2 und 6.3 enthalten eine Sammlung von Maßnahmen, wie Angriffe auf Agenten bzw. Angriffe auf Plattformen verhindert oder zumindest rechtzeitig erkannt werden können. Die aufgezeigten Maßnahmen verfolgen dabei unterschiedliche Ziele, nicht nur darin, ob sie Agent oder Plattform schützen und ob sie Angriffe verhindern oder erkennen, sondern auch darin welche Informationen durch sie geschützt werden. Für Agenten ist es z. B. wichtig den Programmcode, dynamische und statische Daten, ihren Zustand und den Migrationspfad zu schützen. Plattformen legen z. B. darauf Wert, dass ihre Ressourcen nicht durch *Denial-Of-Service*-Attacken blockiert werden, keine Informationen ausspioniert werden und sie ihre Reputation nicht dadurch verlieren, dass Agenten sich auf ihnen gegenseitig attackieren können. Im Abschnitt 6.4 werden die zuvor vorgestellten Maßnahmen einander gegenübergestellt und hinsichtlich ihrer unterschiedlichen Ziele und Möglichkeiten bewertet.

Da Sicherheit auch etwas mit Vertrauen zu tun hat und hinter jedem Programm ein Mensch steckt, werden vorab im folgenden Abschnitt 6.1 noch organisatorische Möglichkeiten diskutiert.

6.1 Organisatorische Maßnahmen

Nicht jede Aufgabe erfordert das höchste Maß an Sicherheit. Schickt man seinen Agenten mit der Aufgabe auf die Reise, möglichst viele Anbieter eines bestimmten antiquarischen Buches zu finden und gegebenenfalls Preise zu erfragen, so ist dieser Auftrag wohl als nicht sehr sicherheitskritisch einzustufen, da der Agent nur Informationen sammelt und keine Kaufverträge abschließt. Es stellt sich die Frage welche Motivation hinter einem Angriff auf diesen Agenten stecken könnte? Trägt der Agent seine gesammelten Informationen nicht mit sich, sondern schickt sie immer sofort nach Hause, kann die aktuelle Wirtsplattform nicht versuchen die Angebote vorangegangener Wirtsplattformen zu manipulieren. Um trotzdem die Wahrscheinlichkeit, den endgültigen Zuschlag zu erhal-

ten, zu erhöhen, könnte sie als Lockmittel einen sehr günstigen Preis angeben oder das antiquarische Buch als vermeintliche Erstausgabe bewerben. Da der eigentliche Kauf des Buches durch den Eigentümer des Agenten stattfindet, wird er diesen Betrugsversuch wohl rasch bemerken. Sollte der Agent am weitermigrieren gehindert werden, wird dies dem Eigentümer mit der Zeit auf Grund der ausbleibenden Nachrichten auffallen und er wird vielleicht einen weiteren Agenten, eventuell mit einer leicht abgeänderten Routenvorgabe, auf den Weg schicken. Im ungünstigsten Fall muss sich der Eigentümer doch noch selbst in das Internet begeben und händisch nach dem gewünschten Buch suchen. In Falle eines derartigen Agenten wäre ein Verzicht auf Sicherheitsmaßnahmen denkbar.

Auch *One-Hop*-Agenten, d. h. Agenten, die nur einen Migrationsschritt durchführen und danach wieder zu ihrer Heimatplattform zurückkehren, sind oft so einfach gestaltet, dass die Wirtsplattform kein offensichtliches Interesse daran haben kann, diese anzugreifen. Der Agent führt keine sensiblen Daten mit sich und die Ausführung des Agenten liegt oft auch im Interesse der Wirtsplattform. [H01]

Will bzw. kann man nicht ganz auf Sicherheit verzichten, so gibt es Ansätze die „vermeintliche" Sicherheit auf Grund organisatorischer Maßnahmen zu erhöhen. Vermeintlich deshalb, weil organisatorische Maßnahmen alleine weder Attacken verhindern noch aufdecken können. Man versucht hierbei potentielle Gefahren auszugrenzen. Sollten diese aber doch einmal, etwa durch die Unachtsamkeit eines Mitarbeiters, eindringen können, ist ihnen das System ohne zusätzliche Maßnahmen schutzlos ausgeliefert.

6.1.1 Vertrauenswürdige Wirtsplattformen (*Trusted Agencies*)

Sind bestimmte Wirtsplattformen von vornherein als vertrauenswürdig einzustufen, kann man mobile Agenten so programmieren, dass sie nur diese vertrauenswürdigen Plattformen besuchen. Entweder indem man ihnen ihre Reiseroute fix vorschreibt oder indem sie ihre nächste Anlaufstation nur aus einem Set von vorgegebenen vertrauenswürdigen Plattformen auswählen dürfen. Diese Maßnahme kann durch das Eingreifen der Wirtsplattformen noch verschärft werden, etwa indem sich die Wirtsplattformen weigern Agenten aufzunehmen, die von einer nicht vertrauenswürdigen Plattform kommen, oder dass sie verhindernd eingreifen, wenn ein Agent versucht zu einer nicht vertrauenswürdigen Plattform zu migrieren. Aglets unterstützt diese Art der *Trust-Based Policy*, hier weigert sich die Wirtsplattform Agenten von oder zu einer Plattform zu migrieren, der sie misstraut.

Um den eben beschriebenen Prozess zu leben sind organisatorische Maßnahmen rund um das Agentensystem nötig. Es sei denn es handelt sich bei dem Agentensystem um einen geschlossenen Bereich, etwa eine Werkshalle ohne Netzanschluss nach außen. In diesem Fall kann man mit größerer Wahrscheinlichkeit davon ausgehen, dass die einzelnen Plattformen vertrauenswürdig sind und bleiben. Anderenfalls benötigt man organisatorische Lösungen, nicht zu letzt etwa die Frage betreffend: Wie informiere ich Agenten und Plattformen darüber, dass eine Wirtsplattform ihre Vertrauensstellung verloren hat?

Wenn es nicht möglich ist den mobilen Agenten am Besuch einer a priori nicht vertrauenswürdigen Plattformen zu hindern, gibt es noch den Ansatz, nur Teile des Agenten auf diese Plattform zu migrieren und die sicherheitskritischen Berechnungen weiterhin auf einer vertrauenswürdigen Plattform

durchzuführen. Hier stellt sich aber die Frage inwieweit ein solches Vorgehen noch mit den eigentlichen Grunddefinitionen eines mobilen Agenten konform geht. Auch werden sich dadurch negative Auswirkungen auf die Performance des Agenten feststellen lassen.

[H01], [BR05]

6.1.2 Soziale Kontrolle

Soziale Kontrolle in einem Agentensystem funktioniert ähnlich dem Bewertungssystem von privaten Online-Verkäufern, z. B. bei eBay oder Amazon. Kunden, die bereits bei einem Anbieter gekauft haben, können hier ihre Erfahrungen anhand von Noten vergeben. Je mehr Kunden positive Erfahrungen mit diesem Anbieter gemacht haben, umso besser wird sein Leumund. Dies geschieht aber auch umgekehrt, je mehr schlechte Erfahrungen mit diesem Anbieter gemacht wurden, umso schlechter wird sein Ruf.

Eine ähnliche soziale Kontrolle ist auch in einem Agentensystem möglich, hier sind die Wirtsplattformen die Anbieter und die Agenten die Kunden, die auf Grund ihrer Erfahrungen mit dieser Plattform Bewertungen vergeben, die an einer zentralen Stelle gesammelt und abgefragt werden können. Betrachtet man den Lebenszyklus dieses Systems geht man aber von Null aus, d. h. zu Beginn gibt es keine Bewertungen und die Agenten müssen den Plattformen a priori vertrauen, erst mit der Zeit werden Erfahrungen gesammelt. Auch bringt diese Art der Vertrauenserlangung die Möglichkeit einer neuen Art von Attacke mit sich, den Rufmord. Ein böswilliger Agent kann eine Wirtsplattform dadurch schädigen, dass er ihr ungerechtfertigt schlechte Noten gibt. Des weiteren kann sich eine böswillige Wirtsplattform lange Zeit still verhalten, gute Noten sammeln und erst dann plötzlich losschlagen, d. h. guten Note kann somit nicht voll vertraut werden. [H01], [BR05]

6.1.3 Verträge

Ein anderer organisatorischer Ansatz wäre die Eigentümer der Wirtsplattformen per Gesetz oder auf Grund von Verträgen zu „zwingen“ die Sicherheit ihrer Plattformen zu garantieren, d. h. die Plattformeigentümer versichern schriftlich, weder Daten oder Code der Agenten auszuspionieren, noch die Agenten sonst auf irgendeine Art und Weise anzugreifen. Des Weiteren versichern sie auch, dass sie die Plattform selbst vor Angriffen von außen schützen werden. [BR05]

Bei diesem Ansatz ist es vor allem wichtig, dass die Plattform die auf ihr durchgeführten Aktivitäten entsprechend aufzeichnet, so dass sie sich im Streitfall darauf berufen kann.

6.2 Schutz von Wirtsplattformen

Wird ein mobiler Agent auf einer Wirtsplattform ausgeführt, sollte die Plattform dafür Sorge tragen, dass ihr Betriebssystem und ihre Ressourcen, wie. z. B. verfügbare CPU-Zeit, Daten auf der Festplatte usw. dabei keinen Schaden nehmen. Im Folgenden werden Maßnahmen aufgezeigt, wie sich

Wirtsplattformen vor böswilligen Agenten schützen können. Diese Maßnahmen können entweder Angriffe verhindern oder zumindest rechtzeitig aufdecken.

6.2.1 Die Sandbox

Java wurde im Laufe der Zeit zu einer der beliebtesten Programmiersprachen für mobile Agentensysteme, sowohl zur Entwicklung der Plattformen als auch der mobilen Agenten selbst. Der Grund dafür sind u. a. ihre Eigenschaften zur Unterstützung mobilen Codes [BR02] und ihre plattformunabhängige Ausführbarkeit.

Ein weiterer Grund für die Verbreitung von Java in mobilen Agentensystemen sind ihre Sicherheitsmechanismen, zu diesen gehört auch die Sandbox. Die Sandbox ist eine Softwaretechnik, die es ermöglicht, Programmcode in einem eigenen begrenzten Bereich auszuführen. Die Grenzen betreffen dabei u. a. Zugriffe auf das lokale Filesystem, Netzwerkzugriffe oder das Aufrufen von lokalen Programmen. Dies soll verhindern, dass ein böswilliger Agent, ob mit Absicht oder unabsichtlich, in seiner Laufzeitumgebung Schaden anrichten kann. Die Sandbox erfordert definierte Sicherheitsrichtlinien (*Security Policy*) für die Ausführung von mobilem Code. Diese Sicherheitsrichtlinien geben Regeln und Grenzen vor, an die sich der mobile Agent halten muss. [AB04]

Ein Java Interpreter enthält drei wichtige Sicherheitskomponenten: *Class Loader*, *Verifier* und *Security Manager*. Der *Class Loader* konvertiert den mobilen Code in Datenstrukturen um, die dann der lokalen Klassenhierarchie hinzugefügt werden. Der *Class Loader* legt auch einen Namensbereich für den mobilen Code an, hier wird sicher gestellt, dass Klassen des mobilen Agenten nicht lokale Klassen der Plattform überschreiben. Der *Verifier* führt, noch bevor der mobile Code geladen wird, diverse Überprüfungen durch, u. a. ob es sich um einen korrekten Java Code handelt, der Stack nicht über- und nicht unterschritten wird oder ob nicht erlaubte Umwandlungen von Datentypen enthalten sind. In den Anfangszeiten von Java, JDK 1.0, wurde streng zwischen *Local* und *Remote* C*lasses* unterschieden, d. h. Klassen der Wirtsplattform und Klassen des mobilen Agenten. Lokale Klassen hatten uneingeschränkte Rechte, die Klassen der mobilen Agenten mussten vor ihrer Ausführung dahingehend überprüft werden, ob sie auch die geforderten Sicherheitsrichtlinien erfüllten, diese Aufgabe übernahm (und übernimmt) der *Security Manager*. Operationen wurden entweder als sicher oder gefährlich eingestuft. Sichere Operationen wurden immer erlaubt, potentiell gefährliche Operationen führten zu einer Exception. Dieses sehr einfache System wurde in der Zwischenzeit deutlich verbessert. Die beiden folgenden Abschnitte, „6.2.2 Signieren des Programmcodes (*Code Signing*)“ und „6.2.3 Zugangskontrolle (*Access Control*)“, beschreiben die darauf aufbauende sicherheitstechnische Weiterentwicklungen von Java. [RG98], [AB04]

6.2.2 Signieren des Programmcodes (*Code Signing*)

Das Signieren des mobilen Programmcodes stellt sicher, dass dessen Integrität gewahrt bleibt. *Code Signing* kann einen Angriff auf den Programmcode zwar nicht verhindern, aber rechtzeitig aufdecken, so dass der veränderte Programmcode nicht zur Ausführung kommt.

Zum Signieren des Programmcodes wird der gesamte Programmcode mit Hilfe einer *One-Way Hash* Funktion einem wesentlich kleineren Wert zugeordnet, der im Anschluss daran mit Hilfe eines asymmetrischen Verfahrens verschlüsselt wird (siehe Abschnitt 3.1.3). Signiert wird der Programmcode des mobilen Agenten entweder durch den Entwickler selbst oder durch den Eigentümer des Agenten. Stammt die Signatur vom Entwickler, so bestätigt dieser dadurch das korrekte Verhalten des Agenten. Stammt die Signatur vom Eigentümer des Agenten, so bestätigt dieser damit lediglich seine Eigentümeransprüche. Die Eigentümersignatur kann nichts darüber aussagen, ob der Agent sich auch wirklich korrekt verhält. Aber auch wenn die Signatur des Programmcodes vom Entwickler stammt, sagt dies noch nichts über die Vertrauenswürdigkeit des Entwicklers aus!

Eine bekannte Implementierung von *Code Signing* ist *Authenticode* von Microsoft, das zur Absicherung von Active X Controls und Java Applets verwendet wird. *Code Signing* ist seit JDK 1.1 auch Teil von Java. Vertraute die Plattform der Signatur, bzw. dem Besitzer des dazugehörigen privaten Schlüssels, wurde der Programmcode in JDK 1.1 mit denselben Rechten wie lokaler Code ausgeführt. Handelte es sich aber um eine nicht vertrauenswürdige Signatur wurde der Code innerhalb einer Sandbox ausgeführt. [AB04]

6.2.3 Zugangskontrolle (*Access Control*)

Java unterstützt seit JDK 1.2 eine Erweiterung des zuvor beschriebenen *Code Signing*. Der Programmcode wird nicht mehr grob in vertrauenswürdig und nicht vertrauenswürdig unterteilt, sondern er wird abhängig von seiner Signatur, bzw. dem Besitzer des dazugehörigen privaten Schlüssels, bewertet. Der Inhaber der Plattform, auf der der Code ausgeführt werden soll, kann nun an jede Identität, d. h. jeder signierenden Einheit, individuelle Rechte vergeben. Zu diesen Rechten gehört z. B. das Vergeben von *Read-*, *Write-*, *Delete-* oder *Execute*-Rechten für einzelne Files oder Verzeichnisse oder das Vergeben von *Connect-*, *Accept-* oder *Listen*-Rechten für bestimmte IP-Adressen oder Ports. Über die Einhaltung dieser Sicherheitsrichtlinien (*Security Policy*) wacht der bereits in Abschnitt 6.2.1 erwähnte *Security Manager*. [NPR06]

6.2.4 Migrationsvergangenheit (*Path History*)

Bei diesem Ansatz zum Schutz der Wirtsplattform führt der Agent eine authentifizierbare Liste über alle von ihm besuchte Plattformen mit sich. Dahinter steckt der Gedanke, dass der Freund meines Freundes nicht unbedingt auch mein Freund sein muss [O96]. Auf Plattformen umgelegt bedeutet dies in etwa, dass die Plattform C zwar der Plattform B vertrauen mag, die davor vom Agenten besuchte Plattform A aber für sie z. B. vollkommen unbekannt, und daher nicht vertrauenswürdig, ist. Je nach dem, wie die aktuelle Wirtsplattform die Migrationsvergangenheit des Agenten bewertet, kann sie entscheiden, ob sie diesen ausführen will, bzw. wenn ja, mit welchen Rechten der Agent während seiner Ausführung ausgestattet wird.

Zur Erstellung einer *Path History* fügt jede Plattform einen signierten Eintrag an die bereits bestehende *Path History* des Agenten an. In diese Signatur fließt die ID der aktuellen Plattform und die ID der nachfolgenden Plattform mit ein. Um nachträgliche Manipulationen zu verhindern, muss auch die Signatur der Vorgänger Plattform mit in die aktuelle Signatur aufgenommen werden

[JK99], [AB04]. Die nachfolgende Plattform kann nun entscheiden, ob sie die mitgelieferte *Path History* des Agenten nur durchsucht oder aber die gesamte Kette für jede Plattform authentifiziert.

Dieses Verfahren kann eine Manipulation des Agenten zwar nicht verhindern, wirkt aber abschreckend, da der Angreifer leicht ermittelt werden kann. Ein Nachteil dieses Verfahrens ist allerdings, dass die mitgeführten Daten des Agenten mit jeder Plattform anwachsen und die zur Authentifizierung des gesamten Pfades benötigte Zeit stetig ansteigt.

In der Literatur findet man die unterschiedlichsten Arten von *Path Histories*. [R98] beschreibt einen Ansatz mit zwei kooperierenden Agenten, wobei jeder Agent die *Path History* des jeweils anderen Agenten überwacht. Will ein Agent migrieren, sendet er über einen authentifizierten Kanal die ID der Vorgänger, der aktuellen und der nächsten Plattform an den kooperierenden Agenten, der diese Informationen wartet und überwacht. Stellt der kooperierende Agent Unregelmäßigkeiten fest, kann er entsprechende Aktionen setzen. Diese Art der *Path History* dient nicht dem Schutz der Plattform, sondern des Agenten, da die Plattform in diesem Fall nicht im Besitz der, für eine Verifikation benötigen, Schlüssel ist. Ein Nachteil dieses Verfahrens ist der zusätzliche Aufwand für den gesicherten Kanal zwischen den beiden Agenten und die Tatsache, dass, sollte ein Agent „sterben“, nicht festgestellt werden kann welche der beiden letzten Plattformen für seinen „Tod“ verantwortlich ist. (siehe Abschnitt 6.3.6) [J99]

6.2.5 Sicherheitsfilter (*Content Inspection*)

SeMoA, die bereits in 5.1.2 vorgestellte Agentenplattform, deren ausdrückliches Ziel schon zu Beginn ihrer Entwicklung die Sicherheit war, beinhaltet in ihrer Architektur auch eine Schicht, die sich der Analyse der Inhalte des Agenten widmet. SeMoAs Aufbau ähnelt dem einer Zwiebel, ein eintreffender Agent muss sämtliche Schichten von außen nach innen hin erfolgreich passieren, bevor er zur Ausführung gelangt. Die äußerste Schicht ist die Transportschicht, die zweite Schicht besteht aus einer Pipeline unterschiedlichster Sicherheitsfilter. Es gibt eine eigene Pipeline für eintreffende und eine eigene Pipeline für ausgehende Agenten. Jeder Filter hat die Macht den Agenten entweder zu akzeptieren oder abzulehnen. Diese Filterprozedur wird in SeMoA auch, in Anlehnung an die von Firewalls bekannten Konzepte, als *Content Inspection* bezeichnet. [RJ01]

Content Inspection kann vereinfacht als schichtweise Aneinanderreihung der zuvor vorgestellten Verfahren betrachtet werden. Die Signatur des Agenten wird überprüft (siehe Abschnitt 6.2.2) und abhängig von seiner Migrationsvergangenheit (siehe Abschnitt 6.2.4) werden Zugriffsrechte (siehe Abschnitt 6.2.3) vergeben.

SeMoAs äußerste Schicht, die Transportschicht verlangt eine gegenseitige Authentifizierung der an der Migration des Agenten beteiligten Plattformen. Bei der Übertragung des Programmcodes selbst wird auf die Wahrung von Integrität und Vertraulichkeit geachtet, z. B. durch Verwendung von SSL (*Secure Sockets Layer*). Abhängig von Sicherheitsrichtlinien kann der Empfang des Agenten verweigert werden. [RJP01]

Des Weiteren enthält SeMoA einen Filter zur Behandlung von digitalen Signaturen, d. h. zur Verifizierung der Signaturen eintreffender Agenten und zur Erstellung der Signaturen ausgehender Agen-

ten. Dabei wird sowohl die Signatur des Eigentümers über den statischen Teil des Agenten geprüft, als auch die Signatur der letzten Wirtsplattform über den gesamten Agenten, d. h. über die statischen und die dynamischen Daten. Ein weiterer Filter ist für die Verschlüsselungen zuständig, d. h. das Entschlüsseln der eintreffenden Agenten und das Verschlüsseln der ausgehenden Agenten. Am Ende der Pipeline für die eintreffenden Agenten gibt es noch einen Filter, der den Agenten mit konfigurierbaren Rechten versieht. Diese Rechte basieren auf den Ergebnissen der zuvor passierten Schichten, können aber zusätzlich auch noch vom Eigentümer des Agenten oder z. B. der ID seiner letzten Wirtsplattform abhängig sein. Die Filter können entweder dynamisch registriert bzw. deregistriert werden oder aber zum Startzeitpunkt des SeMoA-Servers fix vorgegeben werden. Um zu verhindern, dass verschlüsselte Teile von einem Agenten in einen anderen Agenten kopiert und von diesem böswillig verwendet werden (C*ut-And-Paste*-Attacke), fordern die Filter einen Beweis darüber, dass der Agent Wissen über die benötigten Schlüssel besitzt. Für Interessierte sei hier auf [RC01] verwiesen.

6.2.6 Proof Carrying Code

1996 entwickelten G. Necula und P. Lee einen effizienten Algorithmus mit dessen Hilfe die Eigenschaften von Software und im Besonderen die Einhaltung von Sicherheitsrichtlinien überprüft und verifiziert werden kann [PCC_WIKI]. Dieser Algorithmus wird *Proof Carrying Code* (PCC) genannt. Die Grundidee hinter PCC ist, an den Programmcode einen leicht zu überprüfbaren Beweis anzuhängen. Mit Hilfe dieses Beweises kann sicher gestellt werden, dass die Ausführung des Codes keinerlei Sicherheitsrichtlinien der ausführenden Plattform verletzt.

In PCC wird zwischen einem *Code Producer* und einem *Code Consumer* unterschieden. Der *Code Producer*, der den Programmcode zur Verfügung stellt, stellt gleichzeitig auch eine Kodierung des Beweises zur Verfügung. Dieser Beweis bestätigt, dass der Programmcode die vom *Code Consumer* definierten Sicherheitsrichtlinien erfüllt. Der *Code Consumer* ist die Plattform auf der der Agent zur Ausführung kommt. Die Verifikation durch PCC teilt sich in zwei unterschiedliche Phasen, eine schwierige und eine einfache. Die schwierige Phase ist die Generierung des Beweises, die einfache Phase ist die Überprüfung des Beweises.

Wie in [NL98] von Necula und Lee beschrieben, besteht eine PCC Session aus fünf Schritten, die nachfolgend eingehender erklärt werden.

Schritt 1.
Eine PCC Session startet damit, dass der *Code Producer* den Programmcode für den Versand an den *Code Consumer* vorbereitet. Dazu stattet er den Programmcode mit Anmerkungen aus, dies kann manuell oder mit Hilfe eines *Certifying Compilers* geschehen. Die Anmerkungen enthalten Informationen, die dem *Code Consumer* helfen sollen, die sicherheitsrelevanten Eigenschaften des Codes besser zu verstehen. Der *Code Producer* sendet nun den mit den Anmerkungen versehenen Programmcode an den *Code Consumer*.

Schritt 2.
Der *Code Consumer* führt nun eine schnelle, aber detaillierte Durchsicht des erhaltenen Programmes durch. Dies geschieht mit Hilfe eines VCGen, eines *Verification Condition Generators*, der Teil der

Sicherheitspolitik des *Code Consumers* ist. Das Programm VCGen führt zwei Checks durch. Als erstes werden die simpleren Sicherheitseigenschaften des gelieferten Programmcodes überprüft, wie etwa, ob alle Sprünge innerhalb der erlaubten Speichergrenzen stattfinden. Die zweite Aufgabe von VCGen ist, den Programmcode nach Instruktionen zu durchsuchen, deren Ausführung die Sicherheitsrichtlinien des *Code Consumers* verletzen könnten. Wird eine derartige Instruktion gefunden legt der VCGen eine *Verification Condition* fest, d. h. er definiert eine Bedingung innerhalb deren Grenze die Durchführung der Instruktion als sicher gilt. Die Summe aller *Verification Conditions* wird *Safety Predicate* genannt und an den *Proof Producer* gesandt. Der *Proof Producer ist* für die Erstellung des Beweises zuständig.

Schritt 3.
Der *Proof Producer* überprüft nun, ob die Anforderungen des *Safety Predicate* erfüllt werden. Ist dies der Fall, sendet er eine Kodierung des Beweises an den *Code Consumer*. Da der *Code Consumer* dem *Proof Producer* nicht vertrauen muss, kann jeder, somit auch der *Code Producer*, als *Proof Producer* auftreten. In der Praxis ist dies auch häufig der Fall.

Schritt 4.
Der *Code Consumer* überprüft nun den gelieferten Beweis. Dies geschieht mit Hilfe eines *Proof Checker* Programms. Der *Proof Checker* überprüft, ob alle Folgerung innerhalb des Beweises auch durch die, in den Sicherheitsrichtlinien definierten, Regeln abgedeckt sind. Auch stellt der *Proof Checker* sicher, dass der Beweis auch wirklich das in Schritt 2 erstellte *Safety Predicate* behandelt. Dies soll eine Attacke durch umgehen des PCC Systems verhindern, z. B. durch Liefern eines zwar korrekten Beweises, der aber ein wesentlich simpleres *Safety Predicate* als Grundlage hatte.

Schritt 5.
Hat der Programmcode sowohl die Prüfung durch VCGen als auch die Prüfung des Beweises positiv bestanden, ist sicher gestellt, dass er die geforderten Sicherheitsrichtlinien nicht verletzt. Er kann somit, ohne zusätzlich benötigte Laufzeitüberprüfungen, installiert und ausgeführt werden.

Die durch PCC abdeckbaren Sicherheitsanforderungen reichen von Speicherzugriffen, über Typkonversionen, bis hin zu Zugriffskontrolle auf Ressourcen oder zeitlichen Beschränkungen während der Programmausführung. PCC ist nicht auf eine bestimmte Programmiersprache beschränkt und benötigt kein Vertrauensverhältnis zwischen den beteiligten Parteien, in Programmcode und Beweis sind alle benötigten Informationen enthalten. Die vom *Code Consumer* benötigten Programme zur Überprüfung des Beweises sind relativ einfach und arbeiten sehr schnell. PCC gesicherte Agenten benötigen während ihrer Durchführung weniger Zeit als Agenten, deren Sicherheit während der Laufzeit überprüft wird.

6.2.7 Status Beurteilung (*State Appraisal*)

Ein mobiler Agent besteht aus seinem Programmcode und seinem aktuellen Status. Dieser Status ist Ausgangspunkt des Agenten bei seinem Start auf einer neuen Wirtsplattform und somit ein geeignetes Ziel für Angriffe. Ein von Farmer et al. in [FSG96] präsentierter Ansatz sieht vor, den Status des Agenten auf seine Gültigkeit hin zu überprüfen. Die Überprüfung selbst findet nach der Authentifi-

zierung des Agenten und vor der Autorisierung der einzelnen Aktionen des Agenten statt, da die Ergebnisse der *State Appraisal* Funktionen Inputwerte für die Autorisierung der Aktionen liefern.

Am besten lässt sich *State Appraisal* anhand des folgenden Beispiels erklären. Nehmen wir an, der Eigentümer eines Agenten benötigt drei Flugtickets nach Wien, die aber nicht notwendigerweise bei ein und derselben Fluggesellschaft gebucht werden müssen. D. h. die Anzahl der vom Agenten zu buchenden Sitzplätze besteht nicht mehr aus statischen drei Stück, sondern kann vom Agenten, je nach Angebot, dynamisch auf mehrere Fluggesellschaften aufgeteilt werden.

Der Entwickler des Agenten liefert nicht nur den Sourcecode sondern auch eine *Appraisal*-Funktion *max*. Die Funktion *max* beinhaltet Angaben über die maximal erlaubten Genehmigungen, die dem Agenten während seiner Lebenszeit eingeräumt werden dürfen. Für obiges Beispiel könnten dies etwa maximal 5 Flugtickets sein. Der Entwickler signiert das fertige Programm mit seinem privaten Schlüssel. Der Sender fügt an den Agenten noch eine weitere *Appraisal*-Funktion *req* (*request*) an. Die Funktion *req* stellt den aktuellen Zustand des Agenten dar und beinhaltet die Anzahl der Genehmigungen, die der Agent tatsächlich während seiner Laufzeit erhalten soll. Für obiges Beispiel sind dies, zu Beginn der Laufzeit des Agenten, somit noch auf dessen Heimatplattform, genau drei Flugtickets. Der Sender fügt seine ID an den Agenten an und signiert diesen ebenfalls mit seinem privaten Schlüssel. Sobald ein Agent eine Buchung vornimmt, wird die Funktion *req* vor der Migration des Agenten entsprechend adaptiert. Auf der neuen Wirtsplattform des Agenten wird vor der Buchung überprüft, ob *req* eine Teilmenge von *max* ist. Wenn ja, können die Flugtickets gebucht werden, wenn nein, dann liegt ein Manipulationsversuch vor. Auf diese Art kann verhindert werden, dass z. B. statt der gewünschten 3 Stück deutlich mehr, z. B. 100 Stück, Flugtickets gebucht werden, so dass die Fluglinie glaubt ausgebucht zu sein und die Kunden zur Konkurrenz, dem möglichen Verursacher des Schadens, abwandern.

State Appraisal schützt sowohl den Eigentümer des Agenten als auch die Wirtsplattform. Da der aktuelle Status des Agenten sehr komplex sein kann, ist eine große Schwierigkeit dieses Ansatzes geeignete Funktionen zu finden. Auch muss sichergestellt sein, dass eine böswillige Plattform die *Appraisal*-Funktionen nicht analysieren und den Agenten entsprechend manipulieren kann. [BR05]

6.2.8 Historisch bedingte Zugriffskontrolle (*History-Based Access Control*)

Edjlali et al. präsentieren in [EAC99] eine Zugriffskontrolle für mobilen Code, und somit auch für mobile Agenten, die auf der Beobachtung und Aufzeichnung des Programmverhaltens beruht. Je nachdem wie sich das Programm bisher verhalten hat, werden ihm bestimmte Zugriffe erlaubt oder verwehrt.

Eine Voraussetzung für *History-Based Access Control* ist die eindeutige Identifizierbarkeit des Agenten (siehe Abschnitt 3.6), da verhindert werden muss, dass sich ein böswilliger Agent die Rechte eines anderen Agenten aneignet. Wurde z. B. ein Agent bereits einmal auf Grund seines böswilligen Verhaltens terminiert, soll natürlich verhindert werden, dass dieser zu einem späteren Zeitpunkt nochmals zur Ausführung kommt. Mit Hilfe der *History-Based-Access-Control*-Sicherheitsrichtlinien bleibt die Information über derartige Terminierungsereignisse erhalten und wird das erneute Ausführen eines Agenten mit dieser Identität verhindert.

Edjlali et al. teilt die Programme z. B. in Browser und Editor ähnliche Programme ein. Ein Browser ähnliches Programm darf sich z. B. mit einer remote gelegenen Seite verbinden, solange es niemals zuvor versucht hat ein lokales File, dass es nicht selbst erstellt hat, zu öffnen. Ein Editor ähnliches Programm darf lokale Files modifizieren, wenn es diese selbst erstellt hat und niemals zuvor versucht hat sich mit einer remote gelegenen Seite zu verbinden. Trifft eine dieser Bedingungen nicht mehr zu, wird den Sicherheitsrichtlinien entsprechend eine *Exception* behandelt. Dies soll verhindern, dass lokale Daten unerlaubt an nicht berechtigte Dritte gesendet werden.

Durch *History-Based Access Control* können z. B. auch Datenbankzugriffe geregelt werden, etwa in der Art, dass einem Programm immer nur der Zugang zu einer Abfrage erlaubt wird, z. B. Durchführungsdatum und Bezeichnung eines medizinischen Eingriffs einerseits und Name des Patienten und Datum des letzten Besuches andererseits. Jede Abfrage für sich ist relativ harmlos, der Zugang zu beiden ermöglicht aber den Namen eines Patienten mit einer medizinischen Behandlung zu verbinden.

Mit Hilfe der *History-Based Access Control* können auch gewisse Formen von *Denial-Of-Service*-Attacken verhindert werden, etwa dadurch, dass z. B. die Zeitstempel der Verbindungsversuche eines Agenten zu seiner Heimatplattform überwacht werden. Abhängig vom Zeitpunkt seines letzten Verbindungsaufbaus, kann ihm ein erneuter Kommunikationsversuch verwehrt und erst nach Ablauf einer gewissen Zeitspanne wieder erlaubt werden.

Allerdings ist im Umgang mit dieser Methode auch eine gewisse Vorsicht angebracht. Auch wenn z. B. der Agent selbst keine remote Verbindung aufgebaut hat, während er die Daten eines bestimmten Files ausliest, kann er sich diese Daten trotzdem noch einverleiben, indem er sich diese an seinen Code anhängt und damit zur nächsten Plattform migriert. Konspirierende Agenten könnten jeweils eine Datenbankabfrage tätigen und diese zu einem späteren Zeitpunkt miteinander abgleichen. Für obiges Beispiel einer medizinischen Datenbank, würde dies bedeuten, dass beide Agenten gemeinsam nun doch wissen welcher Patienten welche medizinische Behandlung erhielt.

6.2.9 Überwachen der Ressourcen (*Resource Reification*)

Binder und Villazòn widmeten sich in [VB01] dem Problem der Überwachung der von mobilen Agenten in Anspruch genommen Ressourcen. Bei den zu überwachenden Ressourcen unterscheiden sie zwischen physikalischen und logischen Ressourcen. Zu den physikalischen Ressourcen zählen u. a. CPU, Speicher und Netzwerkbandbreit. Zu den logischen Ressourcen zählen z. B. die Anzahl der Threads oder die Anzahl der Agenten. Dazu kommt noch die Überwachung der Kommunikation, um gegebenenfalls regulierend in die Kommunikationsbandbreite oder die Nachrichtengröße eingreifen zu können. Bei der Überwachung der physikalischen Ressourcen legten sie besonderen Wert auf eine betriebssystemunabhängige Realisierung.

Um aus der Überwachung der Ressourcen Vorteile ziehen zu können, muss es Möglichkeiten geben auf deren Ergebnisse entsprechend zu reagieren. Dies kann sowohl in der Laufzeitumgebung, d. h. innerhalb der Plattform, oder im Agenten selbst realisiert werden. Binder und Villazòn kombinierten

beide Ansätze zu einer *Reflective Execution Environment* (REE), einer Laufzeitumgebung, die es ermöglicht, dass das ausgeführte Programm, der Agent, auf Rückmeldungen aus der Überwachung reagieren kann. Die REE besteht aus zwei Schichten: Einer Basisschicht und einer Metaschicht. In der Basisschicht wird der eigentliche Agent ausgeführt, in der Metaschicht wird der dazugehörige Metaagent ausgeführt, der auch auf die Rückmeldungen der Ressourcenüberwachung reagieren kann. Beim Eintreffen des mobilen Agenten auf der Plattform findet eine *load-time transformation* statt, d. h. der eintreffende Agent wird analysiert und modifiziert. Durch diese vor Ort stattfindende Modifikation wird vermieden, dass für unterschiedliche Laufzeitumgebungen unterschiedliche Agentenversionen benötigt werden. Für die Modifikation des Agenten, d. h. das Einbringen der Metaobjekte, wird mobiler Code verwendet, der dynamisch in die REE geladen werden kann. Binder und Villazòn nennen diesen Code *agent modifier*. Ein *agent modifier* kann von einer *Trusted Third Party* angefordert werden. Durch dieses Vorgehen, der Modifikation des Agenten vor Ort, bleibt der mobile Agent selbst unverändert und kann weiterhin auch auf Plattformen ohne REE ausgeführt werden.

Die Aufgaben von Basisschicht und Metaschicht innerhalb der REE zeigen sich am Beispiel der Überwachung der Netzwerkbandbreite. Hier werden die in der Basisschicht getätigten Aufrufe an die entsprechenden Metaobjekte in der Metaschicht umgeleitet. Die Metaobjekte leiten die Aufrufe an die tatsächlichen Komponenten weiter, zeichnen aber gleichzeitig auch die Auslastung auf und können limitierend eingreifen. Speicher- und CPU-Überwachung sind etwas komplizierter, da es hier nicht ausreicht einfach einen eintreffenden Aufruf umzuleiten. Im Falle der Speicherüberwachung muss das Anlegen bzw. auch wieder das Zerstören jedes Objektes überwacht werden, dazu wird dynamisch ein eigener Speicherallokator in den Programmcode eingefügt und alle Konstruktoren und Destruktoren werden entsprechend ersetzt. Bei der Überwachung der CPU entschieden sich Binder und Villazòn zu einem Ansatz basierend auf der Anzahl der vom Agenten durchgeführten Instruktionen. Zwecks Erstellung eines Kontrollflussgraphs wurde der Bytecode des Agenten analysiert und an strategischen Punkten Zähler eingefügt. Diese Zähler wurden auf der Metaebene dazu verwendet um gegebenenfalls regulierend in die Priorität der Threads einzugreifen.

Das Wissen über die verbrauchten Ressourcen bietet die unterschiedlichsten Möglichkeiten. Wie in Abschnitt 5.1.3 beschrieben, zählen *Denial-Of-Service*-Attacken zu den am schwierigsten in den Griff zu bekommenden Attacken. Überwacht eine Wirtsplattform ihre Ressourcen, kann sie sich vor derartigen Angriffen schützen, indem sie entsprechende Gegenmaßnahmen setzt, wie z. B. die Priorität derjenigen Threads, die zu viel CPU verbrauchenden, nach unten zu stufen. Auch kann dem Agenten bzw. dessen Eigentümer für die verbrauchten Ressourcen ein entsprechender Betrag in Rechnung gestellt werden. Kostenpflichtige Anfragen können helfen die Auslastung der Wirtsplattform in Grenzen zu halten, da es im eigenen Interesse des Agenten liegt seinen Auftrag möglichst kostengünstig zu erledigen. Der Overhead für eine derartige Überwachung wird in [VB01] für den Fall einer Speicherüberwachung mit 10% und für den Fall einer CPU-Überwachung mit 20% angegeben. Die Autoren weisen allerdings darauf hin, dass sie keinerlei Optimierung in ihre Realisierung eingebaut haben und daher hier noch Verbesserungen möglich sind.

6.3 Schutz von mobilen Agenten

Mobile Agenten benötigen für ihre Ausführung die Unterstützung einer Plattform. Nicht jede Plattform ist aber hundert Prozent vertrauenswürdig. Auch kann ein Agent Informationen mit sich führen, die nur für bestimmte Plattformen lesbar und für die übrigen Plattformen geheim sein sollen. Der folgende Abschnitt widmet sich den unterschiedlichsten Möglichkeiten wie ein Agent sich, d. h. seinen Programmcode, und seine Daten vor böswilligen Plattformen schützen kann. Unter Schutz ist hier einerseits das Verhindern eines Angriffs, anderseits aber auch das nachträgliche Aufdecken einer erfolgten Attacke zu verstehen, da es nur einige wenige Ansätze gibt, die Angriffe bereits von vornherein verhindern können.

6.3.1 Verschlüsselte Funktionen (*Encrypted Functions*)

Sander und Tschudin stellten in [ST97] und [ST98] ein Verfahren vor, das einzelne Programmteile eines mobilen Agenten verschlüsselt, bei dem aber dennoch die Eigenschaft der Mobilität des Agenten erhalten bleibt, d. h. der Agent kann weiterhin auf fremden Wirtsplattformen ausgeführt werden. Dieses Verfahren ermöglicht es Agenten ihre Funktionen auch in nicht vertrauenswürdiger Umgebung sicher durchzuführen. [BR05]

Der zugrunde liegende Gedanke ist, dass der Agent nur die verschlüsselte Funktion mit sich führt und über die tatsächliche unverschlüsselte Funktion keine Informationen besitzt:

1. Alice besitzt die mathematische Funktion f und verschlüsselt diese.
   Das Ergebnis ist $E(f)$
2. Alice sendet das Programm $P(E(f))$, das $E(f)$ ausführt, an Bob.
3. Bob führt $P(E(f))$ mit dem Eingabewert x aus.
4. Bob sendet $P(E(f))(x)$ zurück an Alice.
5. Alice entschlüsselt $P(E(f))(x)$ und erhält dadurch x.

Zur Realisierung der *Encrypted Functions* schlagen Sander und Tschudin additive und mutiplikative Homomorphismen oder verkettete Funktionen (*Composite Functions*) vor. Ein hybrider Ansatz unter Verwendung beider Verfahren wird in [LAH04] vorgestellt. Grundlage dieses Ansatzes sind der Drei-Adress-Code, ein homomorphes Verschlüsselungsschema und verkettete Funktionen.

Compiler arbeiten bei der Übersetzung des Sourcecodes in ausführbaren Code in Zwischenschritten, manche Compiler generieren auch Darstellungen dieser Zwischenschritte. Der Drei-Adress-Code ist eine dieser Darstellungen. Er ist eine Aneinanderreihung von Befehlen der Form x := y op z, wobei x, y, z für Namen, Konstante oder temporäre Werte stehen und op für einen Operanden. So wird z. B. der Ausdruck x + y * z wie folgt dargestellt:

$t_1 := y * z$
$t := x + t_1$

„Ein Homomorphismus ist eine Abbildung zwischen zwei mathematischen Strukturen, durch die Teile der einen Struktur auf bedeutungsgleiche Teile der anderen Struktur eindeutig abgebildet wer-

den“ [H-WIKI]. Additiv multiplikative Homomorphismen stellen sicher, dass das Ergebnis einer Berechnung mit zwei verschlüsselten Werten gleich dem verschlüsselten Ergebnis derselben Berechnung mit unverschlüsselten Werten ist. Zusätzlich zur additativen und multiplikativen Eigenschaft muss ein homomorphes Verschlüsselungsschema auch mixed-multiplikativ sein, d. h. die Verschlüsselung zweier multiplizierter Werte x*y muss gleich der Multiplikation der Verschlüsselung von x mit dem unverschlüsselten Wert y sein, E(x*y) = E(x) * y. Zur Vermeidung von Anomalien sollte nur ein Element diese zu letzt genannte Eigenschaft erfüllen. Zum Vergleich, bei den Ganzzahlen erfüllt diese Eigenschaft die Zahl 1.

Verkettete Funktionen lassen sich gut am folgenden Beispiel erklären. E, der Eigentümer des Agenten, möchte auf der Plattform P seine Funktion *h(x)* berechnen lassen. Der Wert x kommt dabei von der Plattform P. E möchte die Funktion aber nicht veröffentlichen und sendet eine Funktion *f(x) = g(h(x))* an P. Auf P wird nun die Berechnung von *f(x)* durchgeführt, die Plattform hat dabei keinerlei Information über die eigentliche Funktion *h*, da sie nur in Besitz von f ist. Um zum gewünschten Ergebnis *h(x)* zu gelangen, führt der Eigentümer die inverse Funktion von *g* aus, $h(x) = g^{-1}(f(x))$. Die in [LAH04] verwendeten Beispielfunktionen sind $g(x) = x^3 + 1$ und $g^{-1}(x) = \sqrt[3]{x-1}$.

Lee et al. verwenden zur Realisierung ihres Ansatzes ein Programm das sie *Mobile Agent Encryption* (MAE) nennen. MAE verwendet als Input den Drei-Adress-Code des Compilers. Die Operanden der Befehle werden mit dem homomorphen Verschlüsselungsschema verschlüsselt. Zur Verschlüsselung des Drei-Adress-Codes selbst werden verkettete Funktionen verwendet. Die Entschlüsselung erfolgt genau umgekehrt, dabei muss aber nicht der ganze Agent entschlüsselt werden, sondern nur die gewünschten Ergebnisse.

Der große Vorteil von *Encrypted Functions* ist, dass der mobile Agent seine Berechnungen direkt mit den verschlüsselten Werten durchführt. Als Nachteil ist zu erwähnen, dass sie nicht vor Blackbox-Attacken schützen können, d. h. die ausführende Plattform könnte den Agenten mehrmals hintereinander ausführen und so zu einer Tabelle gelangen, die unterschiedlichen Werten *n* das zugehörige verschlüsselte Ergebnis *E(n)* zuweist. Dadurch könnten Algorithmen, die mit Hilfe solcher Tabellen leicht erkennbar sind, rekonstruiert werden. Laut Sander und Tschudin stellt hierbei die einzige Schwierigkeit die notwendige Größe einer solchen Tabelle dar. Hohl merkt in [H98] noch einen weiteren Nachteil an, er weist darauf hin, dass bei diesem Verfahren Klartextdaten nur an die Heimatplattform, oder eine vertrauenswürdige Wirtsplattform, geliefert werden können, da zur Entschlüsselung dieser Daten der entsprechende Schlüssel benötigt wird. Nicht vertrauenswürdigen Wirtsplattformen können keine Klartextinformationen übermittelt werden.

Zurzeit können mit einem homomorphen Verschlüsselungsschema leider nur mathematische Grundfunktionen, d. h. Additionen und Multiplikationen, geschützt werden. Die Technik ist noch nicht ausgereift genug, um sie für mobile Agenten mit komplexen Aufgaben im Allgemeinen anwenden zu können, das Finden entsprechender Verschlüsselungsfunktionen E erweist sich als äußerst schwierig. Es gibt auch zahlreiche Annahmen und Bedingungen auf die dieser Ansatz aufbaut. Interessierte seien hier auf [LAH04] verwiesen.

6.3.2 Zeitlich begrenzte Blackboxen (*Time-Limited Black Boxes*)

In [H98] bezeichnet Hohl *Encrypted Functions* als eine Methode einen Agenten in eine Blackbox zu verwandeln. Ein Agent ist eine Blackbox, wenn sowohl sein Programmcode als auch seine Daten zu keiner Zeit auslesbar oder modifizierbar sind, lediglich die Eingangs- und Ausgangsdaten des Agenten sind einsehbar. Hohl fügt dieser Definition eine zeitliche Komponente hinzu. Ein Agent ist eine zeitlich begrenzte Blackbox, wenn sowohl sein Programmcode als auch seine Daten innerhalb eines bestimmten Zeitintervalls weder auslesbar noch modifizierbar sind. Nach Ablauf dieses Zeitinterfalls ist ein Angriff auf den Agenten, d. h. auslesen oder modifizieren von Code oder Daten möglich, führen aber zu keinerlei negativen Folgen für den Agenten.

Die bei den zeitlich begrenzten Blackboxen verwendete Methode hat das Ziel, die Analyse des Agenten, d. h. das Ausforschen von Code und Daten, für den Angreifer so schwer wie möglich zu gestalten. Dabei geht es nicht darum, die einzelne Codezeile zu schützen, sondern darum, den Sinn, der hinter allen Codezeilen versteckt ist, d. h. die Semantik des Agenten, zu schützen. Hohl schlägt vor den Code des Agenten so stark zu verändern und zu verwirren, dass der von einem automatisierten Programmanalysator benötigte Zeitaufwand zu hoch wird und für den Angreifer keine erfolgreiche Analyse innerhalb der zeitlichen Grenzen möglich ist. [BR05]

Hohl stellt drei Methoden zum Verändern eines Agenten vor:

Neuanordnen der Variablen (*Variable Recomposition*)

Dieser Algorithmus zerteilt einzelne Variablen in mehrere Stücke und bildet aus diesen unterschiedlichen Stücken neue Variablen. Abbildung 6.1 zeigt ein Beispiel. Die Variablen a, b und c werden zerteilt. Die neuen Variablen v23 und v19 entstehen dadurch, dass sie aus den unterschiedlichen Stücken der original Variablen zusammengebaut werden. Diese Stücke selbst können dabei eine Anzahl von Bits unterschiedlicher Länge sein.

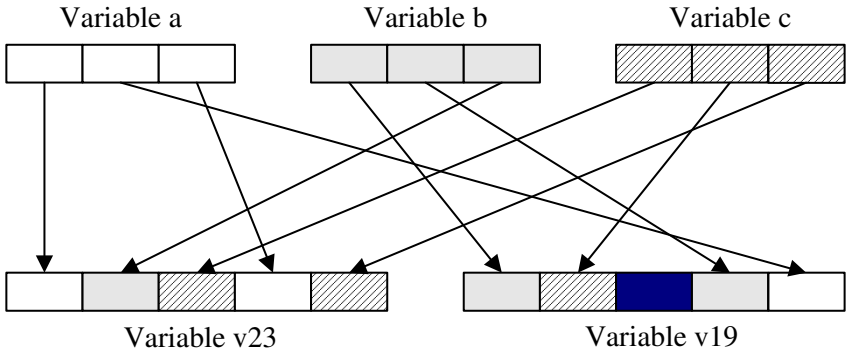


Abbildung 6.1: Neuanordnen der Variablen (*Variable Recomposition*) [H98]

Kontrollflusselemente in laufzeitabhängige Sprünge umwandeln

Um den Ablauf eines Programmes zu verbergen, können bedingten Befehle (*Conditionals*) oder Schleifen (*Loops*) durch werteabhängige Sprünge ersetzt werden, deren Inhalt erst während der Laufzeit bekannt ist. Nachfolgend ein Beispiel aus [H98]:

```
Originalcode:

if (a(b) < c) {
   b = s(d(e) + f); }

Umgewandelter Code:

z = 0
DO
   if (z = 0) then t1 = a(b);   z = 1; continue;
   if (z = 1) then t2 = t1 < c; z = 2; continue;
   if (t2)    then t3 = d(e);   z = 3; continue;
   if (z = 3) then t4 = t3 + f; z = 4; continue;
   if (z = 4) then b = t4;      z = 5; continue;
   if (z = 5) then break;
LOOP
```

Hinterlegen von Schlüsselinformationen (*Deposited Keys*)

Werden Informationen, die für den Ablauf des Agenten wichtig sind, auf eine vertrauenswürdige Plattform ausgelagert, erhöht dies die Sicherheit des Agenten. Ein Angreifer kann seine Analyse nur noch zur Laufzeit des Agenten durchführen. Die ausgelagerten Informationen könnten z. B. die Werte der Variable z im obigen Programmcode sein. z bestimmt welcher Befehl als nächstes ausgeführt wird. Sind diese Werte nicht im Sourcecode enthalten, ist es einem Angreifer nicht möglich, den Agenten vor seiner Ausführung zu analysieren. Die vertrauenswürdige Plattform liefert die Werte für z erst, wenn der Agent den entsprechenden Zustand erreicht hat.

Eine entsprechende Sicherheit wird erst durch die Kombination der eben vorgestellten Methoden erreicht. Die zeitliche Komponente des Agenten wird über ein, von einer CA signiertes, Zertifikat, das das Ablaufdatum des Agenten enthält, realisiert. Die vertrauenswürdige Plattform, die die *Deposited Keys* verwaltet, muss im Besitz dieses Zertifikats sein, da sie nur mit einem Agenten kommunizieren darf, der noch „gültig“ ist. Ein „abgelaufener“ Agent könnte bereits manipuliert worden sein. Der Ansatz der *Time-Limited Black Box* erfordert daher eine Zeitsynchronisation zwischen den beteiligten Komponenten. Kommuniziert der Agent mit einer nicht vertrauenswürdigen Komponente geschieht dies über Tokens. Token sind signierte Dokumente, die ebenfalls ein Ablaufdatum enthalten. Der Empfänger muss die zeitliche Gültigkeit dieser Tokens überprüfen. Ein Agent oder ein Token darf keine Informationen enthalten, die nach Ablauf seiner Gültigkeit noch Schaden verursachen könnten, z. B. darf der maximal erlaubte Kaufpreis, der für alle Agenten desselben Eigentümers Gültigkeit hat, nicht enthalten sein.

Zeitlich begrenzte Blackboxen haben einige Nachteile, dazu gehört unter anderem das Problem das geeignetste Ablaufdatum zu bestimmen. Die Zeit, die eine automatische Analyse des Agenten benötigt, um hinter die Semantik eines bestimmten Codeteiles zu kommen, hängt stark von der Qualität der verwendeten „Verwirrungs"-Techniken und der Erfahrung des potentiellen Angreifers in Programmanalysetechniken ab. [BR05]

Ein Angreifer könnte auch einen klassischen Blackbox-Angriff durchführen, d. h. er führt den Agenten mit unterschiedlichsten Eingangsvariablen aus und beobachtet die dazugehörigen Ausgabewerte bzw. die dabei entstehenden Ablaufmuster. Will der Agent einen Kauf tätigen könnte die Plattform diverse Angebote stellen. Sobald das gestellte Angebot unter dem Limit des Agenten liegt, würde sich das Verhalten des Agenten ändern; sei es nur, dass er sich das Angebot zwischenspeichert, um noch weitere Plattformen zu befragen oder um den Kauf zu tätigen. Um einen derartigen Angriff zu verhindern, schlägt Hohl vor mit einer vertrauenswürdigen Plattform zusammenzuarbeiten und diese immer darüber zu informieren, sobald der Agent ausgeführt wird. So könnte eine zu schnell hintereinander erfolgende oder parallele Abarbeitung des Agenten verhindert werden. Um Rückschlüsse aus dem Ablaufmuster des Agenten zu verhindern sollte der Agent mit Dummycode ausgestattet werden.

Die Existenz des Ablaufdatums macht es unmöglich mit diesem Ansatz einen mobilen Agenten hoher Lebendauer zu entwerfen. Je höher das Ablaufdatum und somit auch die Lebensdauer des Agenten ist, desto höher wird die Wahrscheinlichkeit einer erfolgreichen Attacke. Je kleiner das Ablaufdatum ist, umso unwahrscheinlicher wird ein erfolgreicher Angriff. Gleichzeitig gilt aber auch, je kürzer das Ablaufdatum und die Lebensdauer des mobilen Agenten ist, desto kleiner wird auch seine maximal zurückzulegende Route. Zeitlich begrenzte Blackboxen schränken somit die Mobilität von Agenten ein. [BR05]

6.3.3 Environmental Key Generation

Riordan und Schneier präsentierten 1998 in [RS98] eine interessante Methode zur Aufbewahrung von Geheimnissen in mobilen Agenten, sie nannten ihren Ansatz *Environmental Key Generation*. *Environmental Key Generation* basiert auf der Grundidee, die vom Agenten transportierten Geheimnisse nicht nur vor böswilligen Angreifern, sondern auch vor dem Agenten selbst geheim zu halten. Der Agent erlangt erst nach Erreichung eines bestimmten Zustandes Zugang zu den verschlüsselten Daten, da ihm erst dann der dazu benötigte Schlüssel zur Verfügung steht. Dieser Ansatz ist auf jede Art zu schützender Daten anwendbar, d. h. auch auf den Programmcode selbst.

Ahnungsloser Agent (*Clueless Agent*)

Riordan und Schneier verwenden den Begriff des „ahnungslosen Agenten" (*Clueless Agent*). Ein Agent, der eine verschlüsselte Anweisung mit sich führt, kennt bis zum entsprechenden Zeitpunkt, an dem ihm der Schlüssel zur Entschlüsselung der Anweisung zur Verfügung steht, seinen eigentlichen Auftrag selbst nicht und wird daher als ahnungslos bezeichnet. Zum Auffinden des Schlüssels K wendet der Agent eine vorgegebene Methode zur Durchsuchung seiner Umwelt an. Sobald die Methode zum Erfolg geführt hat, kann der Agent seine Anweisung entschlüsseln. Ist M ein Wert,

den der Agent mit sich führt, und N ein aus der Umwelt des Agenten gelieferter Wert, so können z. B. folgende Methoden zur Gewinnung des Schlüssels K verwendet werden:

- `if hash(N) = M then let K := N`
- `if hash(hash(N)) = M then let K := hash(N)`
- `if hash(`N_i`) = `M_i` then let K := hash(`N_1`,…,`N_i`)`

Nur aus seinem Wissen über M kann der Agent nicht auf den Schlüssel K schließen, da er für die Berechnung von K immer auch eine Information aus seiner Umwelt benötigt.

Zeitstempel und *Environmental Key Generation*

In [RS98] werden auch Möglichkeiten zur Schlüsselgenerierung vorgestellt, die eine zeitliche Abhängigkeit beinhalten. Um *Dictionary*-Attacken zu verhindern wird dabei mit einer TTP (*Trusted Third Party*) zusammengearbeitet. Drei Schritte sind nötig:

1) Der Programmierer erhält von der TTP einen Schlüssel K.
2) Der Programmierer fügt sowohl die mit K verschlüsselten Daten in den Agenten ein, als auch einen Teil der zur Entschlüsselung benötigten Daten und die Information wo der restliche Teil abzuholen ist, d. h. die Adresse der TTP.
3) Während der Laufzeit kontaktiert der Agent die TTP und erhält die noch benötigten Informationen, um die Nachricht entschlüsseln zu können.

Beschränkungen wie „Schlüssel ist erst ab Zeitpunkt T1 verfügbar“ (*Forward-Time Construction*) oder „Schlüssel gilt nur bis zum Zeitpunkt T2“ (*Backward-Time Construction*) sind möglich. Die beiden Möglichkeiten können auch kombiniert werden, so dass der Schlüssel K nur innerhalb eines bestimmten Intervalls, von T1 bis T2, von der TTP zur Verfügung gestellt wird.

Im Folgenden ein Beispiel für die Umsetzung einer *Forward-Time* Hashfunktion:

1) Der Programmierer sendet eine Zielzeit T^* und eine Zufallszahl R an die TTP.
2) Der Programmierer erhält von der TTP folgende Daten, wobei T die aktuelle Zeit und S ein Geheimnis des Servers ist: hash(hash(S, T^*), hash(R, T)).
3) Der Programmierer setzt P = hash(R, T), wobei R eine Zufallszahl ist, und K = hash(hash(S, T^*), hash(R, T)). Die mit K verschlüsselten Daten und P werden in den Agenten eingefügt.
4) Zur Laufzeit kontaktiert der Agent in definierten Abständen die TTP und erhält dabei jeweils ein S_i = hash(S, T_i).
5) Der Agent versucht mit K = hash(S_i, P) = hash(hash(S, T_i), hash(R, T)) seine Daten zu entschlüsseln. Dies gelingt erst, wenn $T_i = T^*$ ist.

Die Verwendung der aktuellen Zeit im Aufbau von P verhindert, dass ein Angreifer die TTP für eine *Dictionary*-Attacke missbrauchen kann. Die Verwendung der Zufallszahl R dient zur Verschleierung von T, um einen *Forward-Time Dictionary*-Angriff gegen den Server zu verhindern.

Ein Nachteil dieses Verfahrens ist, dass eine böswillige Plattform den Agenten nach Erhalt des Schlüssels nach wie vor angreifen kann, so könnte sie z. B. seinen Code modifizieren, so dass der Agent die entschlüsselte Anweisung nicht ausführt, sondern ausdruckt. [JK99] Auch könnte eine böswillige Plattform den Agenten mit falschen Umweltinformationen versorgen und so verhindern, dass er jemals bis zur Ausführung seiner verschlüsselten Anweisung gelangt.

6.3.4 Replizierung der Plattform (*Agency Replication*)

In [S97] stellt Schneider u. a. ein Protokoll vor, wie Angriffe auf Agenten durch Ausführen ein und desselben Agenten auf mehreren gleichen, d. h. replizierten, Plattformen erkannt und elliminiert werden können. Abbildung 6.2 zeigt den Weg des bzw. der Agenten. Die einzigen nicht replizierten Plattformen sind die Heimatplattform A_0 und die Zielplattform A_n, wobei Heimatplattform und Zielplattform meist ein und dieselbe Plattform sind. Die einzelnen Migrationsschritte des Agenten zwischen A_0 und A_n sind in Stufen unterteilt, jede Stufe entspricht einem Migrationsschritt. Die Heimatplattform sendet mehrere Kopien ein und desselben Agenten an mehrere Plattformen der Stufe 1, diese bearbeitet den Agenten und sendet wiederum mehrere Kopien des Agenten an mehrere Plattformen der Stufe 2 usw. Die Plattformen ab der Stufe i = 2 erhalten somit mehrere Kopien von Agenten der Stufe i-1 und müssen über ein Auswahlverfahren entscheiden, welche der eintreffenden Agenten fehlerhaft sind oder korrumpiert wurden. Es wird davon ausgegangen, dass nie mehr als die Hälfte der Agenten attackiert wurden bzw. fehlerhaft sind, daher wird aus der Mehrzahl der Agenten, die z. B. alle denselben Status oder dieselben Ergebnisdaten haben, ein beliebiger Agent ausgewählt und an die Plattformen der nächsten Stufe weitergeleitet. Auf der Zielplattform wird dasselbe Auswahlverfahren angewandt, der daraus resultierende Agent enthält das gewünschte Ergebnis. Schneiders Ansatz basiert somit auf der Annahme, dass die Mehrheit der auf einer Plattform eintreffenden Agenten nicht böswillig bzw. fehlerhaft ist.

Damit eine Plattform vor ihrem Auswahlverfahren die böswilligen Agenten erkennen und ausschließen kann schlägt Schneider ein Protokoll basierend auf Authentifizierungsketten vor, ähnlich der bereits vorgestellten *Path History* (siehe Abschnitt 6.2.4). Dabei wird jedem Agenten der von Plattform A_i zu Plattform A_{i+1} migriert ein *Forward* mitgegeben, eine digitale Signatur, die bestätigt, dass der Agent von Plattform A_i abgeschickt wurde und sein Ziel Plattform A_{i+1} ist, d. h. die IDs der beiden Plattformen fließen in die Signatur mit ein. Die durchzuführenden Schritte auf einer Plattform der Stufe i teilten sich wie folgt auf:

- Empfangen aller Agenten der Vorgänger Stufe i-1.
  Unter den empfangenen Agenten können sich auch böswillig eingeschleuste Agenten befinden.
- Verwerfen aller Agenten, die keine verifizierbare Reihe an *Forwards* mit sich führen.
  Dabei wird vorausgesetzt, dass die Plattform i die Heimatplattform des Agenten a priori, d. h. von vornherein, kennt, denn nur die Heimatplattform ist in Besitz des privaten Schlüssels von A_0, der in den ersten Forward einfließt. *Forwards* von böswilligen Agenten können nicht mit diesem Schlüssel signiert sein.
- Die verbleibenden Agenten werden verglichen und aus der Mehrzahl der gleichen Agenten wird ein Agent für die weitere Migration ausgewählt.

- Bevor der Agent zur Plattform A_{i+1} migriert wird seiner Reihe an *Forwards* der *Forward* des nächsten Migrationsschrittes A_i nach A_{i+1} angehängt. Da die Stufe i+1 aus mehreren Plattformen unterschiedlicher Identitäten besteht, muss dies für jeden Agenten extra erfolgen, jeder Agent hat eine eigene Reihe an *Forwards*.

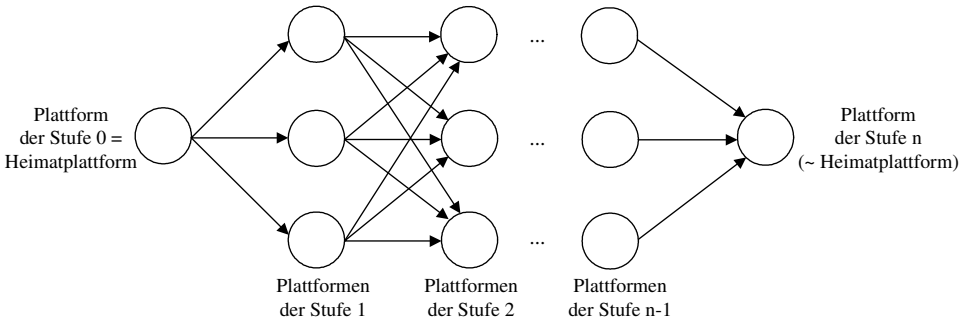


Abbildung 6.2: Replizierung der Plattform [S97]

Yee weist in [Y99] darauf hin, dass der Ansatz, Agenten durch Replizierung der Plattformen zu schützen, unrealistisch ist, da der Ansatz davon ausgeht, dass die einzelnen Plattformen unabhängig voneinander fehlerhafte Ergebnisse liefern. Er weist darauf hin, dass die Plattformen Gemeinsamkeiten wie dieselbe Hard- oder Software oder denselben Administrator haben können. Diese Gemeinsamkeiten machen es möglich, dass alle Plattformen dieselben Fehler liefern können. Läuft auf jeder Plattform die Kopie derselben Software und entdeckt ein Angreifer eine Sicherheitslücke, erhält er dadurch Zugriff auf alle Plattformen des Verbundes. Das Verfahren ist nicht anwendbar auf Plattformen und Agenten, die miteinander Verträge abschließen, wie etwa bei der Reservierung oder dem Kauf eines Flugtickets. Auch schlägt das Verfahren fehl, wenn mehr als die Hälfte aller Agenten fehlerhaft ist, da dann aus der Mehrzahl der Agenten ein fehlerhafter Agent für die weitere Migration ausgewählt wird.

6.3.5 Replizierung des Agenten (*Agent Replication*)

In [Y99] stellt Yee einen Ansatz zur Erkennung eines Angriffs auf mobile Agenten vor, der auf der Replizierung des Agenten beruht. Yees Ansatz funktioniert nur unter bestimmten Nebenbedingungen und erkennt nur die Beeinflussung durch maximal eine böswillige Plattform. Der mobile Agent wird repliziert und durchläuft seine Reiseroute $\{A_0, A_1, \ldots, A_n\}$ einmal in der Richtung von Plattform A_0 nach A_n und einmal in der entgegengesetzten Richtung von A_n nach A_0. Die Nebenbedingungen dabei sind, dass die Route fix vorgegeben und unveränderlich ist.

Im folgendem wird als Beispiel ein Agent verwendet, der die Aufgabe hat das billigste Angebot für ein bestimmtes Produkt zu ermitteln. Befinden sich keine böswilligen Plattformen auf ihrer Route, werden beide Agenten mit dem gleichen Resultat, nämlich Plattform A_i bietet das Produkt am billigsten an, auf ihrer Heimatplattform eintreffen. Ist A_{i-1} eine böswillige Plattform, so wird das Ergebnis des Agenten mit der Route von A_0 nach A_n davon nicht beeinflusst sein, da die böswillige Plattform A_{i-1} zu diesem Zeitpunkt das billigste Angebot der Plattform A_i noch nicht kennt und es somit auch nicht unterbieten kann. A_{i-1} unterbietet mit ihrem Versuch zu betrügen lediglich die teu-

reren Angebote. Der Agent mit der Route von A_n nach A_0 führt, wenn er auf die böswillige Plattform A_{i-1} trifft, allerdings bereits das günstigste Angebot von A_i mit sich. Die Plattform A_{i-1} spioniert dieses aus und unterbietet es. Auf der Heimatplattform A_0 treffen die beiden Agenten mit zwei unterschiedlichen Ergebnissen ein, A_i und A_{i-1}. Yees Ansatz geht davon aus, das Ziel der böswilligen Plattform zu kennen, nämlich das günstigste Angebot zu liefern und dadurch den Zuschlag zu erhalten. Dadurch weiß der Eigentümer der Agenten, dass in diesem Fall ein Agent korrumpiert wurde, und zwar der mit dem billigsten Angebot, d. h. er weiß auch, dass A_i das korrekte Ergebnis ist!

Ist das Ziel der böswilligen Plattform aber z. B. einer bestimmten Konkurrenz zu schaden geht dieser Ansatz nicht auf. Manipuliert die böswillige Plattform z. B. das aktuell günstigste Angebot in der Art, dass es leicht unterboten werden kann, erhält unter Umständen die unbeteiligte dritte Plattform A_{i-2} den Zuschlag, da diese im Anschluss an die böswillige Plattform A_{i-1} das manipulierte Angebot unterbietet. Auf der Heimatplattform treffen wiederum zwei Agenten mit unterschiedlichen Ergebnissen ein, A_i und A_{i-2}. Der Eigentümer der Agenten geht davon aus, dass der Agent mit dem billigsten Angebot A_i korrumpiert wurde und nimmt daher an, dass das Ergebnis der Plattform A_{i-2} das korrekte Ergebnis ist. Sobald die böswillige Plattform andere Ziele als ihren eigenen Vorteil verfolgt schlägt dieser Ansatz somit fehl.

Ist diejenige Plattform, die das billigste Angebot liefern würde, gleichzeitig auch die böswillige Plattform, tritt ein ähnlicher Fall ein. Die Plattform muss nicht ihr wahres Angebot liefern, sondern unterbietet lediglich marginal das aktuell billigste Angebot. Je nach Route (A_0 nach A_n bzw. A_n nach A_0) werden zwei unterschiedliche aktuelle Angebote bei A_i eintreffen, die beide marginal unterboten werden. Hat die böswillige Plattform beide Male das gleiche Angebot gestellt, kann die Heimatplattform keine Manipulation erkennen, erhält aber trotz Manipulation, das günstigste Angebot. Unterscheiden sich die beiden Werte, weiß die Heimatplattform, dass eine Manipulation vorliegt und da beide Angebote von derselben Plattform stammen kennt sie auch den Verursacher.

Wie bereits erwähnt unterstützt dieses Verfahren nur das Auffinden von maximal einer böswilligen Plattform. Gibt es z. B. zwei böswillige Plattformen, die links und rechts von der Plattform A_i mit dem billigsten Angebot positioniert sind, so wird keines der auf der Heimatplattform eintreffenden Ergebnisse korrekt sein! [BR05]

6.3.6 Pfadüberwachung mit kooperierenden Agenten (*Secure Itinerary Recording*)

Plattformen können Agenten auf unterschiedlichste Art und Weisen attackieren, z. B. dadurch, dass sie ihre vorgegebene Route manipulieren oder ihre Migration zur nachfolgenden Wirtsplattform verhindern. Folgender Lösungsansatz von Roth in [R98] zur Routenüberwachung bei mobilen Agenten stützt sich auf zwei voneinander unabhängige Agenten α und β, die nach jeder Migration Informationen über ihre Wirtsplattformen austauschen, d. h. sie teilen einander die ID der Vorgänger Plattform A_{i-1}, die ID der aktuellen Plattform A_i und die ID der nachfolgenden Plattform A_{i+1} mit. Beide mobile Agenten starteten von derselben Heimatplattform A_0 aus. Nachdem der Agent α migriert ist sendet er u. a. die ID seiner aktuellen Wirtsplattform an β. Entspricht diese nicht der von β erwarteten ID A_{i+1} so weiß β, dass es zu einer Attacke kam, kann aber nicht sagen welche Plattform der Übeltäter ist. Entweder hat die Plattform A_i den Agenten α zur falschen Plattform migriert, oder

die aktuelle Wirtsplattform A_{i+1} will der Plattform A_i Schaden und übermittelt dem Agenten α eine falsche Information seine Vorgängerplattform betreffend. [BR05]

Roth weist den Plattformen entsprechend ihrer Vertrauenswürdigkeit Farben zu. Eine weiße Plattform ist voll vertrauenswürdig. Meistens gibt es nur eine weiße Plattform, die Heimatplattform, da diese unter der Kontrolle des Eigentümers des Agenten steht. Plattformen, bei denen die Gefahr besteht, dass sie bei einer Attacke mit mindestens einer anderen Plattform zusammenarbeiten würden, weist Roth die Farbe rot zu. Alle anderen Plattformen erhalten die Farbe grau, dies sind Plattformen die nicht voll vertrauenswürdig sind und möglicherweise böswillig werden könnten. Roths Ansatz zur Routenkontrolle mit Hilfe kooperierender Agenten baut auf folgender Bedingung auf: Keine rote Plattform einer Plattformgruppe A hat Interesse daran mit einer roten Plattform der Plattformgruppe B zusammen zu arbeiten. Daher migrieren die beiden kooperierenden Agenten α und β in unterschiedlichen Plattformgruppen, um sicherzustellen, dass das Protokoll nicht durch miteinander kooperierende Plattformen unterwandert wird. Entscheidend für dieses Protokoll ist, wie diese Bedingung in der Realität erreicht bzw. umgesetzt werden kann. Eine weitere Grundlage für Roths Protokoll ist, dass der Informationsaustausch zwischen den beiden Agenten über einen authentifizierten Kanal stattfindet.

Die Idee hinter diesem Ansatz ist, dass die kooperierenden Agenten nicht nur ihre Routen gegenseitig überwachen, sondern auch, im wahrsten Sinne des Wortes, Geheimnisse miteinander teilen, d. h. wichtige Informationen werden zwischen den beiden Agenten aufgeteilt, so dass sie nicht nur von einer einzigen böswilligen Plattform ausspioniert werden können. Roth bringt als Beispiel Chaums *Digital-Electronic-Cash*-Protokoll, das sicherstellt, dass doppelte Zahlungen erkannt werden und der Betrüger aufgedeckt wird. Details sind in [R98] nachzulesen.

6.3.7 Schutz der statischen Daten (*Static Data Protection*)

Mobile Agenten führen Daten mit sich, die während ihrer gesamten Lebenszeit unveränderlich bleiben. Beispiele für diese statischen, nur auslesbaren und nicht veränderbaren Daten sind etwa der Routenplan, die Adressen der zu besuchenden Plattformen, der Name des Eigentümers des Agenten oder der Name bzw. die ID des Agenten selbst. Bei all diesen Daten ist es wichtig, sicher zu stellen, dass sie nicht böswillig verändert wurden. Dabei kann wie folgt vorgegangen werden: Auf der Heimatplattform wird ein Hash, z. B. SHA-1, über die zu schützenden Daten berechnet. Dieser Hash wird mit dem privaten Schlüssel des Agenteneigentümers signiert, $Sig_{E\text{-}priv}(h(data))$. Nur der Eigentümer des Agenten ist in Besitz dieses privaten Schlüssels, alle übrigen Plattformen auf der Route des Agenten kennen nur den dazugehörigen öffentlichen Schlüssel $K_{E\text{-}pub}$. Bevor der Agent auf die statischen Daten zugreift, verifiziert die Wirtsplattform die Integrität dieser Daten, indem sie einen eigenen Hash über diese Daten rechnet, die mitgelieferte Signatur mit $K_{E\text{-}pub}$ entschlüsselt und die beiden Hashwerte miteinander vergleicht. Stimmt das Ergebnis überein, $h(data) \equiv K_{E\text{-}pub}(h(data))$, kann den Daten vertraut werden. [BR05]

Die Versuche der böswilligen Plattform diesen Schutzmechanismus zu unterwandern sind, solange sie nicht in Besitz von $K_{E\text{-}priv}$ gelangt, zum Scheitern verurteilt. Eine Manipulation der Daten, ohne die Signatur anzupassen, wird von der nachfolgenden Plattform aufgedeckt, da der von ihr berechne-

te Hashwert nicht zum Inhalt der mitgelieferten Signatur passt. Das Auffinden neuer Daten, die zur vorhandenen Signatur passen, gilt als mathematisch unmöglich (siehe Abschnitt 3.1.2) und die Signatur selbst kann von der böswilligen Plattform nicht neu berechnet werden, da nur die Heimatplattform in Besitz von $K_{E\text{-}priv}$ ist. Berechnet die böswillige Plattform die Signatur mit ihrem eigenen privaten Schlüssel, stellt die nachfolgende Plattform fest, dass der verwendete Schlüssel nicht zum Eigentümer des Agenten gehört. Ein Sonderfall ist, wenn die böswillige Plattform den Namen des Agenteneigentümers durch ihren eigenen Namen ersetzt und danach die Signatur mit ihrem eigenen privaten Schlüssel berechnet. In diesem Fall hat sie den Agenten sozusagen gekidnappt und lässt ihn nun für sich, statt für seinen wahren Eigentümer arbeiten. Damit eine solche Attacke erfolgreich sein kann, muss die Empfängerplattform des Agenten in Besitz des öffentlichen Schlüssels des Angreifers sein. Ist dies der Fall, kann die Empfängerplattform einen derartigen Angriff nicht erkennen. Selbst der Einsatz einer PKI, einer *Public Key Infrastructure* (siehe Abschnitt 3.1.4), würde in einem solchen Fall nicht helfen, da der Angriff sozusagen aus dem Inneren des Systems erfolgt. Das Kidnapping eines Agenten lässt sich aber durch das in Abschnitt 6.3.6 vorgestellte Verfahren aufdecken. Zwei kooperierender Agenten α und β informieren sich gegenseitig über ihre letzte, ihre aktuelle und ihre nächste Wirtsplattform. Meldet sich der gekidnappte Agent α nicht mehr bei β oder meldet sich bei β ein Agent mit unbekannter ID, kann β die Heimatplattform über eine Attacke informieren.

6.3.8 Datenzugriff nur für bestimmte Plattformen (*Target Agencies*)

Ein weiterer Ansatz, eine aktive Attacke durch eine böswillige Plattform aufzudecken, ist, die Ergebnisse des Agenten für jede einzelne besuchte Plattform mit einem eigenen Schlüssel von den anderen Plattformen abzukapseln. Die zu einem späteren Zeitpunkt stattfindende Verifikation kann entweder auf der Heimatplattform oder auf definierten Zwischenstationen des Agenten stattfinden. Diese Kapselung kann aus unterschiedlichen Gründen, und daraus resultierend auf unterschiedliche Arten, geschehen. Aus Gründen der Vertraulichkeit werden die Daten verschlüsselt, zur Wahrung der Integrität bzw. zur Festlegung der Verantwortlichkeit werden die Daten mit einer digitalen Signatur versehen. Bei den zu schützenden Daten handelt es sich üblicherweise um dynamische Daten, wie etwa die Ergebnisse der vom Agenten durchgeführter Abfragen.

Es werden drei alternative Möglichkeiten zur Kapselung der Teilergebnisse unterschieden:

- Man stellt dem Agenten alle zur Kapselung benötigten Mittel zur Verfügung.
- Man vertraut darauf, dass die Plattform alle zur Kapselung benötigten Fähigkeiten besitzt.
- Man vertraut auf eine dritte vertrauenswürdige Einheit (*Trusted Third Party*), die die digitalen Fingerabdrücke der Ergebnisse mit einem Zeitstempel versieht.

Keine dieser Möglichkeiten verhindert das böswillige Verhalten einer Plattform, sie ermöglichen aber das Aufdecken bestimmter Attacken. Der Vorteil der ersten Alternative, den Agenten mit allen zur Kapselung benötigten Mittel zu versehen, liegt allerdings in seiner dadurch erlangten Unabhängigkeit zur Plattform. [JK99]

Manche Daten, die ein mobiler Agent mit sich führt, sind nur für die Augen einzelner Plattformen bestimmt, z. B. Daten, die auf der Heimatplattform definiert wurden, aber nicht alle Plattformen der

Route lesen sollen, oder aber Antwortdaten, die nur für die Heimatplattform des Agenten bestimmt sind.

Nachfolgend dazu zwei Lösungsansätze:

Der erste Ansatz wurde von Karnik in [K98] präsentiert. Die zu schützenden Daten werden mit dem öffentlichen Schlüssel $K_{Z\text{-}pub}$ der Zielpattform verschlüsselt, $K_{Z\text{-}pub}(data)$. Da nur die Zielpattform im Besitz des dazugehörigen privaten Schlüssels $K_{Z\text{-}priv}$ ist, sind die Daten für keine andere Plattform lesbar. Zum Sicherstellen der Authentizität der Daten, wird mit Hilfe des privaten Eigentümerschlüssels $K_{E\text{-}priv}$ eine Signatur über den Hash der zu schützenden Daten und dem Namen der Zielpattform ID_Z berechnet, $Sig_{E\text{-}priv}(h(data) + ID_Z)$. Die Zielpattform entschlüsselt die Daten $K_{Z\text{-}pub}(data)$ mit $K_{Z\text{-}priv}$ und die Signatur mit $K_{E\text{-}pub}$. Sie verifiziert, dass es sich bei ID_Z tatsächlich um ihre ID handelt, rechnet über die erhaltenen Daten einen Hash und vergleicht das Ergebnis dieser Berechnung mit dem Hashwert $h(data)$ innerhalb der Signatur. Stimmen die Werte überein kann den Daten vertraut werden.

Der Nachteil dieses Verfahrens ist, dass die Daten für jede Zielplattform extra verschlüsselt werden müssen, d. h. n Zielplattformen bedeuten n Verschlüsselungen. In [RC01] wird ein hybrides Verfahren vorgestellt, dass mit symmetrischen und asymmetrischen Schlüsseln arbeitet. Die zu schützenden Daten werden mit einem symmetrischen Schlüssel verschlüsselt und nur dieser symmetrische Schlüssel wird n mal mit den öffentlichen Schlüsseln der gewünschten Zielpattformen verschlüsselt. Längere Daten können dadurch schneller verschlüsselt werden, da symmetrische Verfahren durchwegs schneller als asymmetrische Verfahren arbeiten.

Beide Verfahren sind anfällig für *Cut-And-Paste*-Angriffe. Eine böswillige Plattform A könnte die für Zielplattform B verschlüsselten Daten in einen anderen Agenten umkopieren. Dieser böswillige Agent reist nun zur Plattform B, lässt dort die Daten entschlüsseln und kehrt mit den entschlüsselten Daten wieder zur böswilligen Plattform A zurück. Auf diesen Weg könnte A in Besitz der Klartextdaten gelangen, die eigentlich nur für B bestimmt waren.

Eine mögliche Lösung für dieses Problem bietet folgender Ansatz aus [RC01]. Der Agent erhält einen statischen Kern, ihm wird ein mit dem privaten Schlüssel $K_{E\text{-}priv}$ des Eigentümers signierter MAC mitgegeben. Der MAC wird über den öffentlichen Teil des Eigentümer Schlüssels $K_{E\text{-}pub}$ und dem symmetrischen Schlüssel k gerechnet, $Sig_{E\text{-}priv}(MAC(K_{E\text{-}pub} + k))$. Die Zielplattform ist bereits in Besitz von $K_{E\text{-}pub}$ und entschlüsselt damit die Signatur. Nun berechnet sie ihren eigenen MAC über $(K_{E\text{-}pub} + k)$ und vergleicht diesen mit dem Inhalt der Signatur. Stimmen die beiden Werte überein, kann die Zielplattform davon ausgehen, dass sie die entschlüsselten Daten dem richtigen Agenten zugänglich macht, da nur der Eigentümer E des Agenten in Besitz des privaten Schlüssels $K_{E\text{-}priv}$ sein darf. Wird nun ein *Cut-And-Paste*-Angriff versucht, erkennt die Zielplattform den Angriff. [BR05]

6.3.9 Schutz der dynamischen Daten (*Dynamic Data Protection*)

Dynamische Daten sind z. B. die Antwortdaten einer Plattform auf die Anfrage des mobilen Agenten. Diese Daten sind für andere Plattformen oder die Heimatplattform bestimmt und müssen, wie die statischen Daten auch, gegen fremde Einsichtnahme und Modifikationen geschützt werden. Sta-

tische Daten können zu ihrem Schutz mit dem privaten Schlüsseln der Heimatplattform signiert und verschlüsselt werden. Dynamische Daten können ähnlich geschützt werden, allerdings werden dabei die Daten mit dem privaten Schlüssel der Wirtsplattform signiert und mit dem öffentlichen Schlüssel der Zielplattform verschlüsselt. [BR05]

Löscht ein Angreifer sowohl Signatur als auch die verschlüsselten Daten oder ändert er die Daten und signiert diese mit seinem eigenen Schlüssel, besteht die Gefahr, dass diese Manipulation nicht auffällt. Um dies zu verhindern, müssen zusätzliche Schutzmaßnahmen vorgesehen werden [BR05], z. B. durch Senden eines Hashwertes über die dynamischen Daten an die Heimatplattform. Die Wirtsplattform berechnet ihren eigenen Hashwert über die erhaltenen dynamischen Daten und lässt diesen Wert von der Heimatplattform bestätigen.

Im Folgenden werden drei Ansätze zum sicheren Transport dynamischer Daten vorgestellt: *Sliding Encryption*, *Append-Only Logs* und *Partial Result Authentication Code*.

Sliding Encryption

Asymmetrische Verschlüsselungsverfahren führen bei kleinen Datenmengen zu einigem Overhead. Jeder Wirtswechsel lässt den Agenten um die Länge der verschlüsselten Antwortdaten samt Overhead anwachsen. Dies kann sich negativ auf seine Migrationseigenschaft auswirken. *Sliding Encryption*, ein in [YY97] präsentierter Ansatz von Young und Yung, ist eine Möglichkeit den Overhead rund um die asymmetrisch verschlüsselten dynamischen Daten zu minimieren. *Sliding Encryption*, ermöglicht es, eine kleine Anzahl an Klartextdaten asymmetrisch zu verschlüsseln und dabei, trotz asymmetrischer Verschlüsselung, auch nur eine kleine Anzahl verschlüsselter Daten als Ergebnis zu erhalten. Dabei wird die kleine Anzahl an Klartextdaten in einem großen Datenblock eingebettet, dieser große Datenblock wird verschlüsselt und ein Teil des verschlüsselten Ergebnisses wird herausgeschnitten und gespeichert. Die nächsten Klartextdaten werden in den verbleibenden, bereits einmal verschlüsselten, Datenblock eingebettet und ebenfalls verschlüsselt, usw. Jede Verschlüsselung ist daher von allen vorangegangenen Verschlüsselungen abhängig.

Im Folgenden wird eine *RSA Based Sliding Encryption* für einen RSA Schlüssel der Länge m, m ist eine Potenz von 2, im Detail erklärt (siehe Abbildung 6.3). Die zu verschlüsselnden Klartextdaten a_1, a_2, ... a_n haben alle eine fixe Länge von u Bytes. Die u Byte Klartextdaten werden vor ihrer Verschlüsselung mit einer Zufallszahl der Länge v Bytes aufgefüllt. Klartextdaten und Zufallszahlen haben eine gemeinsame Länge t, t = u +v, wobei t ein Teiler der RSA Schlüssellänge m ist, t ist auch um ein Vielfaches kleiner als m. Die v Bytes Zufallszahlen übernehmen eine Funktion ähnlich dem eines Initialisierungsvektor IV in einem symmetrischen Algorithmus (siehe Abschnitt 3.1.2). Weiters wird noch ein Stack S, ein Akkumulator A und ein Fenster W benötigt. Die Länge von A und W und die Länge der einzelne Elemente des Stacks entsprechen der Länge m des RSA Schlüssels. Akkumulator und Fenster bestehen aus m/t Elementen und können somit als Arrays der Länge m/t für Elemente der Länge t angesehen werden.

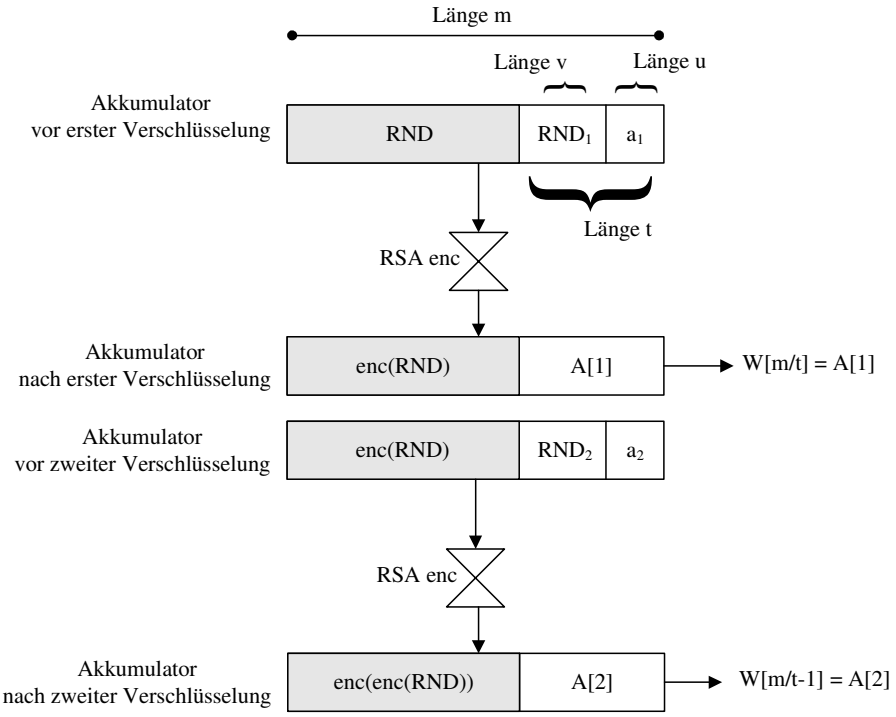


Abbildung 6.3: ***Sliding Encryption***

Zu Beginn der Verschlüsselung ist der Stack leer, der Akkumulator enthält eine Zufallszahl. Die zu verschlüsselnden Daten a1 werden in das *Least-Significant*-Feld des Akkumulators, A[1], gestellt. A[1] besteht nun aus v Byte Zufallszahlen und u Byte Klartextdaten. Der Inhalt des Akkumulators wird nun RSA verschlüsselt. Die *least significant* t Byte des verschlüsselten Ergebnisses werden aus dem Akkumulator A in das Fenster W an die Position m/t geschrieben, W[m/t] = A[1]. Die Versschlüsselung der Klartextdaten a_2 erfolgt ähnlich. Die *Most-Significant*-Daten des Akkumulators bleiben unverändert, d. h. wie sie nach der Verschlüsselung sind. Die *least significant* t Bytes werden mit v Byte Zufallszahlen und u Byte Klartextdaten a_2 gefüllt. Nach der Verschlüsselung des Akkumulators werden die *least significant* t Bytes des verschlüsselten Ergebnisses wieder aus dem Akkumulator A in das Fenster W geschrieben, diesmal allerdings an die Position m/t-1. Sobald die Verschlüsselung von $a_{m/t}$ in W[1] steht wird der gesamte Inhalt von W auf den Stack S gepusht. Die folgenden Werte $a_{m/t+1}$ bis $a_{2m/t}$ werden gleich berechnet und ebenfalls auf den Stack gepusht, usw.

Die Entschlüsselung erfolgt in genau umgekehrter Reihenfolge. Die zu letzt in das Fenster W eingefügten Daten werden in den Akkumulator geschrieben und dieser entschlüsselt. Die *least significant* t Bytes des Akkumulators enthalten das Klartextergebnis. Sobald das Fenster W leer ist werden die obersten Daten aus dem Stack geholt und in das Fenster geschrieben. Die Entschlüsselung wird solange fortgesetzt bis der Stack geleert ist.

Sliding Encryption birgt die Gefahr in sich, dass eine böswillige Plattform Teile des Akkumulators oder des Stacks löschen könnte. Wird der Akkumulator verändert oder gelöscht, sind alle Ergebnisse des Agenten verloren, da sie nicht mehr entschlüsselt werden können. Werden Teile des Stacks ma-

nipuliert oder gelöscht, sind nur die Daten bis zu dieser Position innerhalb des Stacks entschlüsselbar, da alle folgenden Daten von den vorangegangenen abhängig sind.

Append-Only Logs

Ein weiterer Ansatz zum Schutz dynamischer Daten in mobilen Agenten stammt von Karnik [K98]. Karnik führt den Begriff der *Append-Only Logs* ein. *Append-Only Logs* sind Container zur Aufbewahrung von Daten. An diese Container können immer weitere Daten angefügt werden. Wird der Inhalt des Containers manipuliert fällt dies auf.

Das Protokoll basiert auf einer auf der Heimatplattform mit dem privaten Schlüssel des Eigentümers signierten Zufallszahl, $S_0 = Sig_{priv\_E}(RND)$. Die Signatur wird dem Agenten auf seiner Reise mitgegeben. Die Zufallszahl selbst muss auf der Heimatplattform geschützt aufbewahrt werden, da sie, sobald der Agent auf die Heimatplattform zurückkehrt, zur Verifikation des gesamten Protokolls dient. Je Plattform i wird die vorhandene Signatur S_{i-1} erweitert, $S_i = K_{pub\_E}(S_{i-1} + Sig_i(data) + ID_i)$. Die von der Plattform i signierten Daten werden an den Container, die Signatur S_{i-1}, angehängt, ebenso die ID der Plattform i. Diese Daten werden mit dem öffentlichen Schlüssel des Eigentümers K_{pub_E} verschlüsselt und bilden den neuen Container S_i. Kehrt der Agent auf die Heimatplattform zurück, so kann diese den Container Schicht für Schicht auspacken und verifizieren. Ist eine der enthaltenen Signaturen nicht verifizierbar wurden der Agent attackiert. Das letzte Ergebnis muss wieder die Zufallszahl RND sein. [BR05]

Mit diesem Ansatz kann die Integrität der dynamischen Daten geschützt werden. Sollen die Daten auch vertraulich bleiben, können sie zusätzlich mit dem öffentlichen Schlüssel des Eigentümers verschlüsselt werden. Ein Nachteil dieses Verfahrens ist, das nur der Eigentümer des Agenten die Daten verifizieren kann. Das Protokoll ist auch angreifbar für *Cut-And-Paste*-Attacken. Ein Angreifer A kann den aktuellen Container in einen eigenen böswilligen Agenten umkopieren und diesen zur Plattform B schicken. Auf Plattform B werden die von A zuvor definierten Werte an den Container angefügt. Danach kehrt der Agent wieder zur Plattform von A zurück. A ist nun in Besitz eines Containers mit von A vorgegebenen Daten, die aber von B signiert wurden. Dieser Container wird von A wieder in den ursprünglichen Agenten eingefügt. Diese Attacke kann auf der Heimatplattform des Agenten nicht erkannt werden. [R01b]

Partial Result Authentication Code (PRAC)

Ein weiterer Ansatz zum Schutz dynamische Daten in mobilen Agenten ist *Partial Result Authentication Code*, kurz PRAC genannt. PRAC wurde von Yee in [Y99] vorgestellt. Bei diesem Ansatz wird je Plattform mit einem eigenen geheimen Schlüssel ein *Message Authentication Code*, kurz MAC genannt, über die Teilergebnisse des Agenten berechnet.

Der Agent wird auf seiner Heimatplattform mit einer Liste geheimer Schlüssel ausgestattet. Je besuchter Plattform wird über die Ergebnisse des Agenten, unter Verwendung eines Schlüssels aus der Liste, ein MAC berechnet. Die Daten über die der MAC berechnet wurde und der MAC selbst ergeben gemeinsam den PRAC. Der PRAC kann sofort an die Heimatplattform weitergeleitet werden oder aber an den Agenten angehängt und mit ihm migriert werden. Der verwendete Schlüssel wird

im Anschluss an die Berechnung, noch vor der Migration zur nächsten Plattform, verworfen und aus der Liste der mitgeführten geheimen Schlüssel entfernt. Das Zerstören der Schlüssel stellt die Vorwärtsintegrität (*Forward Integrity*) der Daten sicher, d. h. keine später besuchte Plattform kann die Ergebnisse vorangegangener Plattformen manipulieren, ohne dass dies zum Zeitpunkt der Verifikation auffällt. Die Verifikation wird von der Heimatplattform, die in Besitz aller geheimer Schlüssel ist, durchgeführt.

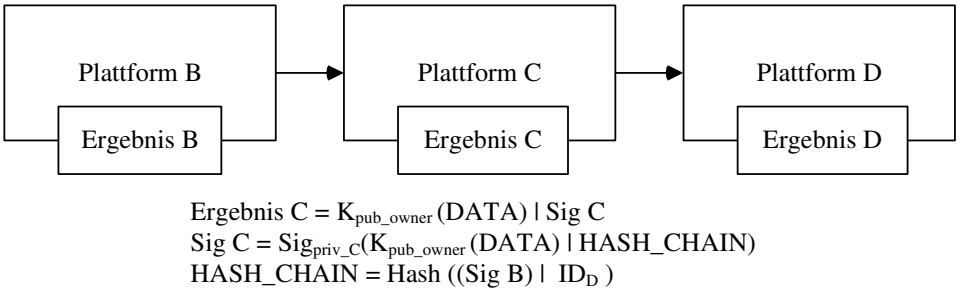
Die PRAC Methode hat einen gravierenden Nachteil. Wird der geheime Schlüssel oder die zur Schlüsselgenerierung verwendete Funktion nach Gebrauch nicht zerstört, sondern von einer böswilligen Plattform aufbewahrt, so kann damit das Ergebnis dieser Plattform zu einem späteren Zeitpunkt manipuliert werden. Diese Manipulation findet statt, wenn die Plattform vom selben Agenten nochmals besucht wird oder aber, wenn der Agent eine Plattform besucht, die mit besagter böswilliger Plattform zusammenarbeitet. Da der Agent bei seinem zweiten Besuch auf der böswilligen Plattform mehr Informationen mit sich führt, z. B. im Falle der Suche nach dem günstigsten Flugticket die entsprechenden Angebote der im Anschluss besuchten Airlines, kann er diese unterbieten, indem er sein zuvor abgegebenes Angebot manipuliert. Eine derartige Manipulation würde bei der Verifikation der Daten auf der Heimatplattform nicht auffallen. PRAC wird zum Schutz der Integrität und nicht der Vertraulichkeit der Daten angewandt. [JK99] Kommt im obigen Beispiel aber auch eine Methode zur Verschlüsselung der Daten zum Einsatz, so kann die böswillige Plattform zwar weiterhin ihr bereits zuvor abgegebenes Angebot manipulieren, jedoch fehlt ihr, um die Konkurrenz zu unterbieten, das Wissen über deren Angebote. Yee schlägt vor die Daten mit dem öffentlichen Schlüssel des Eigentümers des Agenten zu verschlüsseln.

Ein Verfahren, das den sicheren Transport dynamischer Daten von einer Plattform zur nächsten gewährleistet, bietet „starke“ Vorwärtsintegrität (*Strong Forward Integrity*), wenn der eben beschriebene Nachteil nicht eintreten kann, d. h. eine Plattform, auch wenn sie mehrmals besucht wird, ihre zuvor gelieferten Ergebnisse nicht manipulieren kann, ohne das dies bei der Verifikation auffällt. Bei der folgenden von [KAG98] vorgestellten Lösung werden die Berechnungen nicht vom Agenten selbst, sondern von der Plattform durchgeführt. Um die Vertraulichkeit der Daten sicher zu stellen, werden diese mit dem öffentlichen Schlüssel des Agenteneigentümers K_{pub_owner} verschlüsselt. Das Ergebnis dieser Verschlüsselung wird, gemeinsam mit einer „Kette von Hashwerten“ mit dem privaten Schlüssel der Plattform digital signiert. Eine „Kette von Hashwerten“ bedeutet, dass der Hashwert der zuvor besuchten Plattform in die Berechnung des aktuellen Hashwertes mit einfließt. Diese „Kette von Hashwerten“, die in die Signatur mit eingeht, verbindet die Ergebnisse der vorangegangenen Plattformen mit der Identität der als nächster zu besuchenden Plattform. [AB04]

Abbildung 6.4 zeigt den Migrationspfad eines Agenten von B nach D und die Berechnung der Ergebnisdaten des Agenten für Plattform C[5]. DATA bezeichnet das Klartextergebnis des Agenten auf der Plattform C, das mit dem öffentlichen Schlüssel der Heimatplattform verschlüsselt wird. Die in die Signatur einfließende „Kette von Hashwerten“ (HASH_CHAIN) ist eine Hash-Berechnung über die Signatur der vorangegangenen Plattform B (Sig B) und der ID der nachfolgenden Plattform D (ID_D). Über die verschlüsselten Ergebnisdaten und der HASH_CHAIN wird mit dem geheimen

[5] Auf die Darstellung des Agenten selbst wurde zwecks Übersichtlichkeit bewusst verzichtet.

Schlüssel der Plattform K_{priv_C} die Signatur Sig C berechnet. Das Ergebnis des Agenten auf Plattform C besteht aus den verschlüsselten Daten K_{pub_owner}(DATA) und der Signatur Sig C, beide werden vor der Migration des Agenten an seine dynamischen Daten angefügt.



Ergebnis C = K_{pub_owner} (DATA) | Sig C
Sig C = Sig_{priv_C}(K_{pub_owner} (DATA) | HASH_CHAIN)
HASH_CHAIN = Hash ((Sig B) | ID_D)

Abbildung 6.4: ***Strong Forward Integrity***

Wird nun eine Plattform von ein und demselben Agenten ein zweites Mal besucht, so kann diese ihre zuvor gelieferten Ergebnisse nicht manipulieren, ohne dass diese Manipulation auffällt. Bezeichnen wir B als die böswillige Plattform und C als diejenige Plattform, zu der der Agent nach seinem Besuch von B migriert ist. Da die Kette von Hashwerten auch aus den Ergebnissen aller voran gegangenen Plattformen besteht, ist in das Ergebnis von C auch der Hashwert über das Ergebnis von B mit eingeflossen. Würde die Plattform B nun zu einem späteren Zeitpunkt ihr erstes Ergebnis E1 durch ein Ergebnis E2 ersetzen, würde bei der Verifikation des Ergebnisses von C auffallen, dass der in der Signatur der Plattform C enthaltene Hashwert nicht verifiziert werden kann.

Die auf der Plattform C berechneten Ergebnisdaten wurden mit dem privaten Schlüssel von C signiert. Zur Verifikation auf der Heimatplattform wird diese Signatur mit dem öffentlichen Schlüssel von C entschlüsselt. Der darin enthaltene Hashwert über die signierten Daten wird von der Heimatplattform nachberechnet und mit dem entschlüsselten Wert verglichen. Dieser Vergleich schlägt fehl, da C für ihre Hash Berechnung den Wert E1 verwendet hat und die Heimatplattform zum Zeitpunkt der Verifikation aber mit dem nachträglich manipulierten Wert E2 rechnet.

Die drei im Abschnitt 6.3.9 vorgestellten Methoden zur Absicherung des Transports dynamischer Daten unterscheiden sich in ihren Zielen und Gefahren. *Sliding Encryption* unterstützt die asymmetrische Verschlüsselung (Vertraulichkeit) kleiner Datenmengen. Die verschlüsselten Daten selbst können nur vom Besitzer des dazugehörigen privaten Schlüssels entschlüsselt werden. Auf diese Weise können auch Daten, die für andere Wirtsplattformen und nicht für die Heimatplattform bestimmt sind, geschützt werden. Zerstört ein Angreifer aber auch nur einen kleinen Teil des Stacks oder des Akkumulators sind alle Ergebnisse verloren. *Append-Only Logs* sichern die Integrität der dynamischen Daten, diese Integrität kann aber nur vom Eigentümer des Agenten bestätigt werden. *Append-Only Logs* sind daher nur bedingt für die Übermittlung dynamischer Daten zwischen Wirtsplattformen geeignet, da ein Angriff auf diese Daten erst nach Rückkehr des Agenten auf seine Heimatplattform festgestellt werden kann. Sie zeigen sich auch anfällig für *Cut-And-Paste*-Angriffe. Durch Anwendung der Erweiterung des *Partial Result Authentication Code* Verfahrens von Karjoth et al. wird sowohl die Vertraulichkeit als auch die Integrität der Daten sichergestellt und *strong for-*

ward integrity garantiert. In [KAG98] stellt Karjoth auch einen Ansatz vor, der die Verifizierung der dynamischen Daten während der Laufzeit des Agenten und nicht erst bei dessen Rückkehr zur Heimatplattform ermöglicht. Eine Übermittlung dynamischer Daten zwischen den Wirtsplattformen wird hierdurch möglich.

6.3.10 Aufzeichnen der durchgeführten Programmschritte (*Execution Tracing*)

Vigna präsentierte in seiner Doktorarbeit [V98] einen Ansatz, der es dem Eigentümer eines Agenten ermöglicht die vom Agenten durchgeführten Schritte nachvollziehen und überprüfen zu können. Der Eigentümer ist dadurch in der Lage, Code oder Kontrollflussmanipulationen am Agenten aufzudecken. Vigna stellte seinen Ansatz unter dem Titel *Cryptographic Traces* vor, in der Literatur findet sich das Verfahren unter der Bezeichnung *Execution Tracing*.

Vignas Ansatz baut auf der Verwendung von digitalen Signaturen auf. Jede Plattform und jeder Agent besitzen ein eigenes Schlüsselpaar. Eine Verschlüsselung der Daten m mit dem privaten bzw. öffentlichen Schlüssel der Einheit A wird im Folgenden mit $K_{A\text{-}priv}(m)$ und $K_{A\text{-}pub}(m)$ dargestellt, Signaturen mit diesen Schlüsseln werden als $\mathrm{Sig}_{\mathrm{A\text{-}priv}}(m)$ und $\mathrm{Sig}_{\mathrm{A\text{-}pub}}(m)$ dargestellt. Die öffentlichen Schlüssel aller beteiligten Komponenten sind untereinander bekannt. $H(m)$ ist der Hashwert über die Daten m. Ein mobiler Agent besteht zum Zeitpunkt i aus seinem Programmcode C und einem Zustand S^i.

Eine Voraussetzung für *Execution Tracing* ist, dass der Programmcode des Agenten statisch ist, d. h. keine Objekte dynamisch zur Laufzeit geladen werden. Vigna unterteilt den Programmcode in weiße und schwarze Anweisungsschritte. Weiße Anweisungen verändern den Zustand des Programmes auf Grund interner Variablen. Die Anweisung $x := y + z$ ist eine weiße Anweisung, wenn y und z interne Programmvariablen sind. Stammen die Werte von außerhalb des Programmcodes, handelt es sich um eine schwarze Anweisung. Ein Beispiel einer schwarzen Anweisung wäre *read(x)*, x ist ein von der Plattform gelieferter Wert. Die Aufzeichnung der durchgeführten Programmschritte des Agenten, der Trace, besteht aus einer Folgen von Paaren *(n, s)*, wobei n eine eindeutige Identifizierung des Programmschritts ist und s eine Signatur. Bei einer schwarzen Anweisung enthält die Signatur s die neuen Werte der internen Variablen. Würde die Plattform auf *read(x)* die Antwort 3 liefern, wäre s die Signatur über *x:=3*. Bei weißen Anweisungen bleibt die Signatur leer.

Migration des Programmcodes von Plattform A nach Plattform B

Soll der Programmcode C von Plattform A nach Plattform B migriert und dort ausgeführt werden, sendet A an B eine Nachricht, die aus den folgenden Komponenten besteht:

$$A,\ K_{B\text{-}pub}(C),\ K_{A\text{-}priv}\,(H(C),\ B,\ A',\ t_A,\ i_A)$$

- A - Die ID der Plattform A.
- $K_{B\text{-}pub}(C)$ - Der mit dem öffentlichen Schlüssel der Plattform B verschlüsselte Programmcode. Da nur die Plattform B den dazugehörigen privaten Schlüssel besitzt ist auch nur sie in der Lage den Programmcode zu entschlüsseln.

- $K_{A\text{-}priv}$ *(H(C), B, A‘,* t_A, i_A*)* - Diese Komponente besteht aus fünf Teilen, die mit dem privaten Schlüssel der Plattform A verschlüsselt wurden. Da B die ID der Plattform A kennt, weiß sie, dass diese Komponente mit dem öffentlichen Schlüssel von A zu entschlüsseln ist. Der entschlüsselte Inhalt dieser Komponente besteht aus:
  1) *H(C)* - Der über den gesendeten Programmcode berechnete Hashwert von A. Die Plattform B berechnet einen eigenen Hashwert über den bereits zuvor empfangenen Programmcode *C* und vergleicht ihr Ergebnis mit *H(C)*. Stimmen beide Werte überein, kann B davon ausgehen, dass der Programmcode nicht verändert wurde und auch wirklich von A abgesendet wurde, da nur A in Besitz des privaten Schlüssels $K_{A\text{-}priv}$ ist.
  2) *B* - Die ID der Plattform B. So kann die Plattform B sichergehen, dass die Nachricht auch wirklich für sie bestimmt war.
  3) *A‘* - Die ID der Plattform, der B die Empfangsbestätigung über den Erhalt der gesamten Nachricht schicken soll. A‘ kann sich mit der ID von A decken oder aber eine vertrauenswürdige dritte Plattform sein.
  4) t_A - Ein Zeitstempel, um Aktualität zu garantieren.
  5) i_A - Eine eindeutige ID der Nachricht, um *Replay*-Attacken zu verhindern.

Als Antwort sendet B an *A‘* folgende Nachricht:

$$B,\ Sig_{B\text{-}priv}(A,\ K_{A\text{-}priv}\ (H(C),\ B,\ A‘,\ t_A,\ i_A))$$

Die Plattform A’ kann an Hand dieser Nachricht verifizieren, dass die Plattform B zum Zeitpunkt t_A von der Plattform A einen Aufruf zur Ausführung des Programmcodes *C* erhalten hat. Anhand der mitgelieferten ID A‘ sieht die Plattform A‘, dass A sie als Empfänger der Bestätigungsnachrichten von B angegeben hat, da $K_{A\text{-}priv}$ *(H(C), B, A‘,* t_A, i_A*)* die ursprünglich von A gesendeten und von A verschlüsselten Daten sind, die B nur mit seinem eigenen Schlüssel $K_{B\text{-}priv}$ nochmals signiert und an A‘ weitergeleitet hat. Ist eine Bestätigung des Hashwertes *H(C)* gewünscht, kann die Plattform A‘ diese Bestätigung jederzeit von Plattform A anfordern.

Nach Abhandlung dieses Protokolls führt die Plattform B nun den Programmcode C aus, dabei entsteht der Trace T_C. Sobald der Agent seine Arbeit auf Plattform B beendet hat sendet diese die folgende Nachricht an die Plattform A‘:

$$B,\ Sig_{B\text{-}priv}(H(S^l),\ H(T_C),\ i_A)$$

Diese Nachricht enthält einen Hash über den Endzustand des Agenten ($H(S^l)$), einen Hash über den Trace ($H(T_C)$) und wiederum den eindeutigen Wert i_A. Der Trace T_C selbst verbleibt auf der Plattform B.

Neben der oben angegebenen Nachricht an die Plattform A‘ sendet B auch die folgende Nachricht an die Plattform A:

$$B,\ Sig_{B\text{-}priv}(K_{A\text{-}pub}(S^l),\ i_A)$$

Die Nachricht enthält den Endzustand des Agenten (S^l), d. h. die Daten, die das Ergebnis des Auftrags des Agenten repräsentieren. S^l kann nur durch die Plattform A entschlüsselt werden, da es mit $K_{A\text{-}pub}$ verschlüsselt wurde.

Bringt A dem von B gelieferten Ergebnis kein Vertrauen entgegen kann es dieses überprüfen indem es von B den Trace T_C anfordert und von A‘ den dazugehörigen Hashwert. Als erstes rechnet die Plattform A einen eigenen Hashwert über den Trace und vergleicht das Ergebnis mit dem von A‘ gelieferten Wert. Stimmen beide Hashwerte überein führt A eine Simulation der Ausführung des Agenten auf der Plattform B durch, dabei verwendet sie die im Trace gelieferten externen Eingabewerte. Die Simulation sollte als Ergebnis den Endzustand S^1 liefern. Ist dies nicht der Fall kann A beweisen, dass die Plattform B eine Manipulation am Agenten durchgeführt hat.

Einer der Nachteile dieses Verfahrens wurde bereits angesprochen, der Programmcode des Agenten muss statisch sein, dadurch wird die Verwendung von Optimierungsmethoden wie just-in-time Kompilierung verhindert. Auch geht der Ansatz davon aus, dass Agenten sich keinen gemeinsamen Speicherbereich teilen und single-threaded sind. Ist dies nicht der Fall müssten während der Überprüfung die Traces aller Agenten, die den gemeinsamen Speicher in Verwendung hatten, verifiziert werden. Im Fall von multi-threaded Agenten müssten die Traces mit Zeitstempeln versehen sein, um die genaue Abfolge innerhalb der einzelnen Threads nachvollziehen zu können. Der dadurch entstehende Overhead würde das Verfahren unbrauchbar machen. Ein weiterer Nachteil ist die Größe der Traces, die trotz Komprimierungsmaßnahmen noch sehr groß werden können. Es bleibt auch offen wie verhindert werden kann, dass eine böswillige Plattform den Trace manipuliert oder die Kopie eines Traces einer anderen Plattform verschickt. [G03]

6.4 Zusammenfassung

Im Kapitel 6 wurden unterschiedlichste Methoden zum Schutz eines mobilen Agentensystems präsentiert. Die gewünschten Sicherheitsziele und die Methoden diese zu erreichen lassen sich dabei in Gruppen unterteilen:

- Es gibt organisatorische Lösungsansätze, die ohne Kryptographie auskommen.
- Es gibt technische Lösungsansätze, die sich kryptographischer Grundelemente bedienen.

Organisatorische Lösungen mögen für kleinere Anwendungen ausreichen, vertrauensvolle Sicherheit können sie allerdings nie bieten. Die technischen Lösungsansätze unterscheiden sich durch ihre Ziele und Umsetzungsmethoden. Die Ziele „Schutz der Plattform“ und „Schutz des mobilen Agenten“ wurden getrennt betrachtet. Die Methoden zum Schutz der Plattform unterscheiden sich darin, welche Teile der Plattform geschützt werden sollen: Ressourcen, Daten, andere auf ihr laufende Agenten, usw. Die Methoden zum Schutz des mobilen Agenten unterscheiden sich darin, welche Teile des Agenten geschützt werden sollen: statischen Daten, dynamische Daten, der Migrationspfad, Programmcode, Kontrollfluss usw.

Nur wenige Methoden können einen Angriff von vornherein verhindern, die meisten Ansätze zum Schutz mobiler Agentensysteme haben als Ziel einen Angriff rechtzeitig zu erkennen und so den im System entstandenen Schaden so gering wie möglich zu halten; nicht jedes Verfahren bietet dabei auch die Möglichkeit an, den Angreifer zu identifizieren.

Ein entscheidender Unterschied zwischen Plattform und Agent ist, dass eine Plattform im Zweifel die Ausführung eines Agent verweigern kann; ein Agent besitzt diese Option nicht, er ist auf die Zusammenarbeit mit der Plattform angewiesen.

Keine der vorgestellten Methoden ist dazu geeignet alleine angewandt zu werden. Sie alle bauen darauf auf, dass die von ihnen nicht abgedeckten Sicherheitsanforderungen durch die Einbeziehung weiterer Protokolle und Methoden abgedeckt werden. Erst eine Kombination unterschiedlichster Ansätze kann Plattform und Agent einen gewissen Grad an Sicherheit gewähren.

Tabelle 2 liefert eine Übersicht über die wichtigsten Ziele der vorgestellten Methoden.

Tabelle 2: Übersicht der Gegenmaßnamen [angelehnt an BR05]

Gegenmaßname	Komponente, deren Sicherheit erhöht wird	Art des Schutzes	Objekte, die geschützt werden
Sandbox	Plattform	Vorbeugend	Laufzeitumgebung der Plattform
Code Signing	Plattform	Erkennend	Programmcode des Agenten
Access Control	Plattform	Vorbeugend	Daten, Programme und Ressourcen der Plattform
Path History	Plattform	Erkennend	Daten und Programmcode d. Agenten
Content Inspection	Plattform	Erkennend	Daten und Programmcode d. Agenten
Proof Carrying Code	Plattform	Vorbeugend	Daten und Programmcode d. Agenten
State Appraisal	Plattform	Erkennend	Daten des Agenten
History-Based Access Control	Plattform	Vorbeugend	Daten der Plattform
Resource Reification	Plattform	Vorbeugend	Ressourcen der Plattform
Encrypted Functions	Agent	Vorbeugend	Daten und Programmcode d. Agenten.
Time-Limited Black Boxes	Agent	Vorbeugend	Daten und Programmcode d. Agenten
Environmental Key Generation	Agent	Vorbeugend	Daten und Programmcode d. Agenten
Agency Replication	Agent	Erkennend	Daten und Programmcode d. Agenten
Agent Replication	Agent	Erkennend	Daten und Programmcode d. Agenten
Secure Itinerary Recording	Agent	Erkennend	Daten und Programmcode d. Agenten
Static Data Protection	Agent	Erkennend	Daten und Programmcode d. Agenten
Target Agencies	Agent	Erkennend	Daten des Agenten
Dynamic Data Protection	Agent	Erkennend	Daten des Agenten
Execution Tracing	Agent	Erkennend	Daten und Programmcode d. Agenten

7. Implementierung des CARE-Sicherheitsmodules

„Vertrauen ist gut,
Kontrolle ist besser!“
(Redewendung)

Im Folgenden wird an Hand eines konkreten Beispiels diskutiert, wie einige, der in Abschnitt 3.2 vorgestellten, Sicherheitsanforderungen in einer an FIPA angelehnten Agentenplattform implementiert werden können. Ziel ist es, die sichere Kommunikation zwischen zwei Agenten bzw. Plattformen zu ermöglichen, d. h. folgende Sicherheitsanforderungen sollen erfüllt werden:

- Authentizität
  Die kommunizierenden Komponenten sollen in der Lage sein, ihren Gesprächspartner eindeutig identifizieren zu können.

- Vertraulichkeit
  Die Klartextdaten sollen vor den Augen unbefugter Dritter geschützt sein.

- Integrität
  Wurden die Daten während ihres Transportes manipuliert, soll dies für den Empfänger der Nachricht erkennbar sein.

Bei der Agentenplattform, in die obige Sicherheitsanforderungen integriert wurden, handelt es sich um CARE. CARE steht für *C Agent Runtime Environment* und ist eine in der Programmiersprache C geschriebene Agentenplattform, die im Rahmen des EU-Projektes PABADIS PROMISE entwickelt wurde und auf dem FIPA-Standard aufbaut. CARE Agenten sind keine mobilen Agenten im eigentlichen Sinn; es sind stationäre Agenten, die Zeit ihres Lebens auf derselben Plattform verweilen. Trotzdem sind sie mobil, d. h. die Laufzeitumgebung des Agenten, die Plattform, befindet sich auf einem RFID-Chip, der in einer Fertigungsanlage eingesetzt wird, während des gesamten Fertigungsprozesses einem Produkt zugeordnet bleibt und mit diesem von Fertigungsstation zu Fertigungsstation transportiert wird, d. h. physikalisch mobil ist. Die Kommunikation in CARE wurde bisher ungeschützt abgewickelt. Da das Ziel dieser Implementierung die Einführung entsprechender Sicherungsmethoden in der Kommunikation zwischen zwei Plattformen bzw. Agenten ist, kann die Tatsache, dass es sich bei CARE um kein mobiles Agentensystem im herkömmlichen Sinn handelt, ignoriert werden.

Auf die Realisierung weiterer Sicherheitsanforderungen, wie z. B. einer Zugriffskontrolle, wurde verzichtet, da in CARE nur ein Agent zur Laufzeit unterstützt wird und dieser sich auf seiner Heimatplattform befindet.

7.1 PABADIS PROMISE

Da CARE im Rahmen des EU-Projektes PABADIS PROMISE entwickelt wurde, soll dieses kurz vorgestellt werden. PABADIS PROMISE ist die Abkürzung für „*Plant Automation based on Distributed Systems; Product oriented Manufacturing Systems for Re-Configurable Enterprises*" und widmet sich der Entwicklung einer Steuerungsapplikation, deren Ziel es ist, den üblichen zentralisierten Ansatz durch die Verwendung verteilter Intelligenz zu ersetzen. Zur Umsetzung dieses Ziels werden in allen drei Schichten der Automationspyramide (*Enterprise Resource Planning* (ERP)/Unternehmensebene, *Manufacturing Execution System* (MES)/Betriebsleitebene und Feldebene) Agenten eingesetzt. Es wird zwischen *Product Agents* (PA), *Resource Agents* (RA) und *Plant Management Agents* (PMA) unterschieden.

Auf Feldebene kontrollieren *Resource Agents* einzelne Fertigungseinheiten und stellen dem System deren Fähigkeiten zur Verfügung; eine derartige Fertigungseinheit kann z. B. ein Roboter oder eine Transporteinheit sein. Der *Resource Agent* verbleibt auf der Fertigungseinheit. Ein Broker verwaltet die gesamten Möglichkeiten der RAs und stellt diese den *Product Agents* zur Verfügung. Die *Product Agents* repräsentieren die Kundenaufträge und sind logisch und physikalisch über RFIDs einem Werkstück zugeordnet. Da die Werkstücke innerhalb der Anlage mobil sind, sind auch die PAs, zumindest physikalisch, mobil. *Plant Management Agents* repräsentieren Funktionen, die nicht dezentralisierbar sind und deren Aufgaben nicht von *Resource Agents* oder *Product Agents* übernommen werden können. [BT06]

In PABADIS PROMISE kommen weiters spezielle RFIDs zum Einsatz. Agenten, die auf diesen RFIDs eingesetzt werden, müssen bestimmte Rahmenbedingungen erfüllen. Die RFID-Technologie und ihre Rahmenbedingungen für Agenten soll im Folgenden näher beschrieben werden.

RFID - Radio Frequency Identification

Der Begriff RFID kann mit „Identifizierung mit Hilfe von elektromagnetischen Wellen" übersetzt werden. RFID ist ein Verfahren zur Identifizierung und Lokalisierung von Gegenständen. Ein RFID-System besteht aus dem RFID-Transponder und einem Lesegerät. Man unterscheidet zwischen passiven und aktiven RFIDs. Passive RFIDs können nur auf Anfragen eines Lesegeräts antworten und nicht selbst aktiv werden. Aktive RFIDs können auch von sich aus eine Kommunikation starten; eine Eigenschaft, die sie besonders für den Einsatz in Prozessleitsystemen interessant machen. Um von sich aus eine Kommunikation starten zu können, besitzen aktive RFIDs eine eigene unabhängige Energieversorgung. Passive RFIDs erhalten ihre Energie aus dem elektromagnetischen Feld der Lesegeräte und haben daher eine relativ geringe Kommunikationsreichweite. Die möglichen Entfernungen eines aktiven RFIDs zum Lesegerät können von wenigen Metern bis zu 100m reichen, im letzteren Fall muss allerdings auf Grund der Funkeigenschaften Sichtkontakt zum Leser bestehen.

Einfache RFIDs können, ähnlich einem Barcode, nur eine simple Nummer (ID) liefern, komplexere RFIDs dagegen besitzen eigene Mikrokontroller, teilweise sogar mit kryptographischem Co-Prozessor, und können mehrere Kilobyte an Daten speichern, aktive RFIDs sogar bis zu mehreren Megabyte. [RFID-WIKI], [BT06]

Die mit dem Werkstück verbundenen RFIDs in PABADIS PROMISE enthalten nicht nur Produktinformationen und Produktdaten, auf ihnen ist auch die Agentenplattform CARE und der *Product Agent* gespeichert. Damit der Agent auch von sich aus eine Kommunikation starten kann, kommen in PABADIS PROMISE aktive RFIDs mit eigener Stromversorgung zum Einsatz. Diese Stromversorgung ermöglicht es auch, dass der PA während seines gesamten Lebenszyklus am RFID aktiv bleiben kann.

Aufgrund der, auf RFIDs herrschenden Ressourcenknappheit, war die Verwendung einer bereits existierenden, komplett FIPA-konformen, Plattform nicht möglich [TSR08]. Man entschied sich daher für die Eigenentwicklung CARE, die im nächsten Abschnitt 7.2 vorgestellt wird.

7.2 CARE – C Agent Runtime Environment

CARE ist eine Agentenlaufzeitumgebung, die die aktive Ausführung genau eines Agenten zur Laufzeit unterstützt, d. h. je RFID sind maximal eine Plattform und ein Agent aktiv. CARE wurde speziell für die Anwendung auf RFIDs optimiert. Da sie ohne Interpreter arbeitet, wie z. B. Plattformen, die die JVM verwenden, ist dies ein entscheidender Vorteil, die Verarbeitungsgeschwindigkeit betreffend. Gleichzeitig wurde auf arbeitsintensive Funktionen, die auf einem RFID-Chip mit nur einem aktiven Agenten nicht benötigt werden, verzichtet. [TSR08]

In FIPA kommunizieren Agenten untereinander über eine eigene Sprache, der sogenannten *Agent Communication Language* (ACL). Die einer ACL-Nachricht zugrunde liegende FIPA-Struktur wurde in CARE als C-Struktur dargestellt und ist im Anhang A abgebildet. Der einzig verpflichtende Parameter dieser Struktur ist das Feld *performative*, das die eigentliche Handlung, die ausgeführt werden soll, beschreibt. Von FIPA vordefinierte Werte finden sich in [FIPA02b]. Mit dem Schlüsselwort „request“ kann z. B. ein Sender den Empfänger dazu auffordern eine bestimmte Aktion auszuführen. Im anschließenden Beispiel fordert der Sender, dass der Empfänger das File db.txt für nachfolgende Lesezugriffe zu öffnet.

```
(request
:content
"open \"db.txt\" for input")
```

Abbildung 7.1 zeigt den Aufbau einer FIPA *Transport-Message*. Die links zu sehende ACL-Nachricht wird entsprechend den Vorgaben kodiert. In CARE kann diese Kodierung entweder, wie in obigem Beispiel dargestellt, in String-Form oder in XML-Form sein. Dabei wird der Nachricht nichts hinzugefügt, sie erhält lediglich ein anderes Aussehen. Die codierte Nachricht wird in FIPA *payload* bezeichnet. Diese *payload* erhält nun noch einen Umschlag, *envelope* genannt. Der *envelope* enthält u. a. Daten, die eine eindeutige Adressierung innerhalb des jeweiligen Transportsystems er-

möglichen. *Payload* und *Envelope* gemeinsam bilden die *Transport-Message;* die Nachricht, die letztendlich verschickt wird. [FIPA02a], [S07]

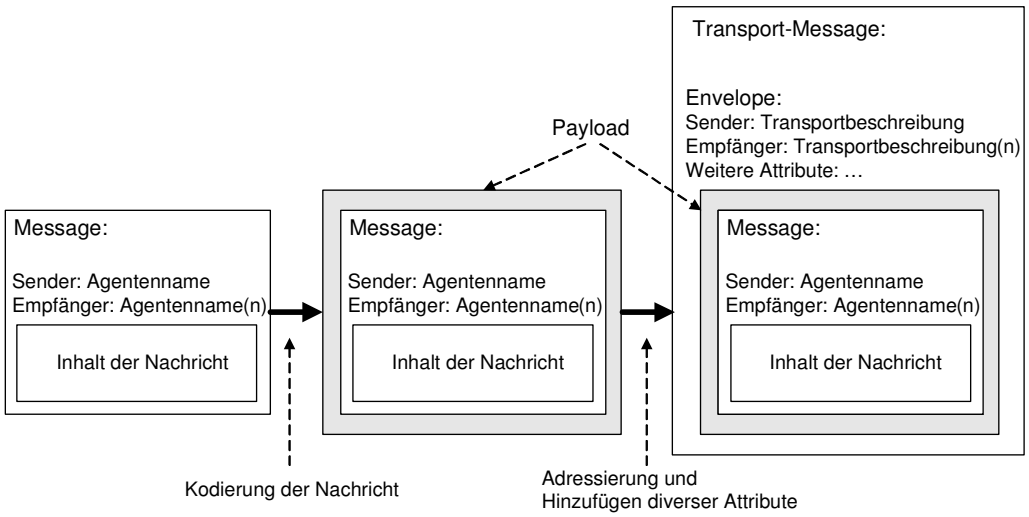


Abbildung 7.1: Aufbau einer FIPA *Transport-Message* [FIPA02a]

CARE wurde als C-Bibliothek implementiert, die von der Applikation des Agenten aus angesprochen wird. Bei Aufruf des Programms wird die Plattform, wie in [S07] beschrieben, initialisiert. Zuerst wird ein sogenannter C-Agenten-Thread gestartet, der die Plattform verwaltet. Dieser Thread kreiert der Reihe nach einen Sender- und einen Empfänger-Thread, die gemeinsam den ACC, den Agent Communication Channel, bilden. Das Message Transport Service, MTS, ist zwecks ressourcensparendem Design in dem Sender- und Empfänger-Thread integriert. Das MTS dient der plattformübergreifenden Kommunikation. Durch die Verwendung des FIPA-konformen http-Protokolls ist CARE in der Lage auch mit anderen Plattformen, wie z. B. JADE, zu kommunizieren. Die Nachrichten selbst werden über eine TCP-Verbindung versendet.

Im Anschluss an die ACC-Threads kreiert der C-Agenten-Thread die, vom Anwender der Applikationsschnittstelle programmierten, *Behaviour*-Threads. Die *Behaviour*-Threads spiegeln das „Verhalten“ des Agenten wider, wird ein Agent z. B. von einer entfernten Plattform oder einem anderen Agenten zur Durchführung einer Aufgabe aufgefordert, muss die Lösung dieser Aufgabe zuvor vom Anwender in einem entsprechenden *Behaviour*-Thread programmiert worden sein.

Auf Codemobilität und die Realisierung eines *Agent Management Systems* wurde in CARE auf Grund der Ressourcenknappheit auf dem RFID-Chip verzichtet. [TSR08]

7.3 FIPA Security Technical Committee

Der auf der „*FIPA Security Technical Committee*“ Webseite angegebene Link auf eine externe Seite mit weiteren Dokumenten und Besprechungsprotokollen zum Thema Sicherheit ist leider nicht korrekt, vermutlich wollte man auf folgende Seite der Queen Mary Universität London verweisen: http://www.elec.qmul.ac.uk/staffinfo/stefan/fipa-security/. Diese Seite trägt ebenfalls die Überschrift

„*FIPA Security Technical Committee*“ und wurde am 24.10.2003 von S. Poslad, dem Vorsitzenden des Sicherheitskomitees, zuletzt aktualisiert. Auf dieser Seite findet sich ein Link auf das *preliminary* Dokument „*FIPA Agent Message Envelope Security Proposal*“ [FIPA03]. Dieses Dokument enthält noch Kommentare und offene Fragen und verweist u. a. auf ein weiteres Dokument „*FIPA X-Slot Security Language Specification Proposal*“, das nicht aufzufinden ist.

[FIPA03] widmet sich dem Thema der Absicherung von Nachrichten und verfolgt dabei genau jene Sicherheitsziele, die zu Beginn dieses Kapitels als Implementierungsvorgabe definiert wurden: Authentizität, Vertraulichkeit und Integrität. [FIPA03] stellt drei mögliche Ebenen vor, die für eine Implementierung der Kommunikationssicherung geeignet wären:

- Applikationsebene
  Eine Kommunikationssicherung auf Applikationsebene entspricht einem proprietären Ansatz und ist laut FIPA nicht zu empfehlen, da die Interoperabilität dadurch verloren geht. Nur Agenten derselben Applikation könnten miteinander gesichert kommunizieren. (siehe Abbildung 7.2 Pfeil 1)

- Transportebene
  Eine Kommunikationssicherung auf Transportebene, als MTS, hat den Vorteil für die kommunizierenden Agenten vollkommen transparent zu sein. Allerdings handelt es sich dabei nur um eine Absicherung der Plattform-zu-Plattform Kommunikation und nicht auch der Agent-zu-Agent Kommunikation. (siehe Abbildung 7.2 Pfeil 2)

- ACL-Ebene
  Eine Kommunikationssicherung auf ACL-Ebene ermöglicht eine sichere Agent-zu-Agent Kommunikation. Die Schutzmaßnahmen können direkt auf die ACL-Nachricht angewandt werden, jede Nachricht kann durch eigene Parameter geschützt werden und der Ansatz ist von der Applikation und dem verwendeten Transportmedium unabhängig. (siehe Abbildung 7.2 Pfeil 3)

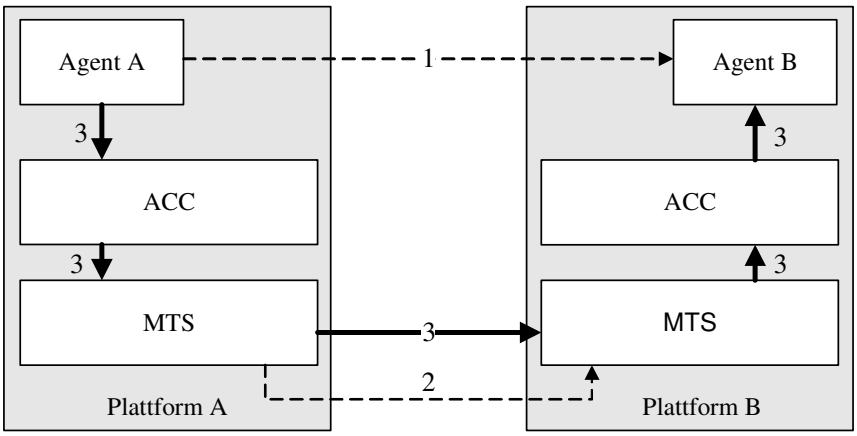


Abbildung 7.2: Sicherungsebenen

Der für die Absicherung der Kommunikation in CARE gewählte Ansatz entspricht dem Schutz auf ACL-Ebene, der auch in [FIPA03] genauer definiert wird. Die Verschlüsselung wird dabei auf die bereits codierte Nachricht, von FIPA auch als *payload* bezeichnet, angewandt (siehe Abbildung 7.1). Der *envelope* der Transportnachricht erhält als Erweiterung sämtliche Daten, die vom Empfänger der Nachricht benötigt werden, um aus der verschlüsselten *payload* wieder eine in Klartext vorhandene ACL-Nachricht zu generieren, diese Erweiterung wird *security-object* genannt.

Es gibt zwei Möglichkeiten, wie diese Erweiterung in den *envelope* integriert werden kann:

1. Die vorhandene FIPA-Struktur kann um einen Eintrag erweitert werden.
   Durch diese Implementierung würde aber die Rückwärtskompatibilität der Struktur verloren gehen.
2. Das *security-object* wird unter dem bereits in *envelope* vorhandenem Feld *user-defined* angefügt.
   In [FIPA03] wird darauf hingewiesen, dass dieser zweite Ansatz möglicherweise vom FAB, dem *FIPA Architecture Board*, abgelehnt werden könnte, da diese Lösung nicht dem vorgesehenen Zweck des *user-defined* Feldes entspricht. FAB ist jene Stelle innerhalb von FIPA, die sicher stellt, dass die Ergebnisse der einzelnen Arbeitsgruppen auch untereinander konsistent sind.

Wie die Erweiterung in den *envelope* integriert wird, ist hauptsächlich eine Angelegenheit der Standardisierung und der Namenskonvention und betrifft nicht die technische Effizienz. Für die Sicherheitsimplementierung in CARE wurde, um mit bestehenden FIPA-Implementierungen kompatibel zu sein, im Rahmen dieser Arbeit der zweite Ansatz gewählt und das *security-object* dem *user-defined* Feld untergeordnet. (siehe Anhang A)

Das im Rahmen dieser Arbeit entworfene CARE *security-object* besteht aus folgenden Elementen:

- *type*
  Das Feld *type* ist ein String und definiert die Art der kryptographischen Operation, die das Objekt *security-object* definiert. Zurzeit werden zwei *type* Werte unterstützt: „SIGN“ und „ENCRYPT“. „SIGN“ bezeichnet eine Signatur, „ENCRYPT“ eine Verschlüsselung.
- *format*
  Das Feld *format* ist ein String und beschreibt die Syntax der folgenden Daten, d. h. den Aufbau des Feldes *security-info*. Das CARE *security-object* verwendet das von [FIPA03] vorgegebene Format „fipa-1“.
- *security-info*
  Der Aufbau der Komponente *security-info* kann der **Tabelle 3** entnommen werden und hängt von den vorangehenden Feldern ab.

Tabelle 3: Aufbau der Komponente *security-info*

Parameter vom Typ String	Beschreibung
algorithm	Der Parameter *algorithm* beschreibt den Prozess, der zum Verschlüsseln der Daten bzw. zum Signieren verwendet wurde. In der aktuellen Implementierung wird RSA_CBC3DES und RSA_WITH_SHA1 unterstützt.
key-ref	Der Parameter *key-ref* enthält die ID des zu verwendenden Schlüssels.
key-size	Der Parameter *key-size* enthält einen BCD-kodierten String, der auf die verwendete Schlüssellänge verweist.
data	Im Fall von *algorithm* = RSA_WITH_SHA1 enthält *data* die Signatur über *payload*. Im Fall von *algorithm* = RSA_CBC3DES enthält *data* den RSA-verschlüsselten symmetrischen Einmalschlüssel.

Daten, die als ASCII-String am Bildschirm dargestellt werden können, entsprechen nach einer Verschlüsselung, im wahrsten Sinn des Wortes, einer Ansammlung kryptischer Zeichen, die auch nicht mehr als String ansprechbar sind. Dieser Umstand könnte bei der Übertragung Probleme mit den darunterliegenden Transportprotokollen verursachen. Aus diesem Grund werden in der Implementierung in CARE die verschlüsselten Daten Base64 kodiert.

Base64 wird u. a. im Dokument RFC4648 [J06] beschrieben und ist ein Verfahren, das 8-Bit lange Binärdaten in eine Zeichenfolge umwandelt, die nur aus wenigen codepage-unabhängigen ASCII-Zeichen besteht. Base64 wird z. B. im Internet-Standard MIME (*Multipurpose Internet Mail Extensions*) beim Versenden von E-Mail-Anhängen angewendet. In MIME ermöglicht Base64 den problemlosen Transport beliebiger Binärdaten, da SMTP (*Simple Mail Transfer Protocol*) ursprünglich nur für den Versand von 7-Bit-ASCII-Zeichen ausgelegt war. Base64 formatierte Daten benötigen etwa um ein Drittel mehr Speicherplatz als die ursprünglichen Daten. [BASE64-WIKI]

Die ursprünglichen Daten werden bei der Umwandlung in 6 Bit Blöcke unterteilt. Mit Hilfe von 6 Bits können maximal die Zahlen 0 bis 63 dargestellt werden. Jede der 64 möglichen Zahlen wird einem codepage-unabhängigen Zeichen zugeordnet. Der String „foobar“ sieht Base64 kodiert z. B. wie folgt aus: „Zm9vYmFy“.

7.4 Signatur und Verschlüsselung in CARE

Eine Nachricht kann in CARE gleichzeitig signiert und verschlüsselt werden, dazu werden mehrere *security-object*s definiert. Die C-Struktur securityObject enthält einen Pointer auf die nachfolgenden Objekte. Abhängig davon, welches *security-object* zuvor definiert wurde, werden entweder die Klartextdaten oder die bereits verschlüsselten Daten signiert.

Signatur

Zum Signieren der Nachricht, d. h. zum Signieren der *payload*, füllt der Agent ein *security-object* mit den gewünschten Werten. Dies kann z. B. wie folgt aussehen:

```
type = "SIGN"
format = "fipa-1"
algorithm = "RSA_WITH_SHA1"
key-ref = "AGENT_KEY1"
```

Zuerst wird die ursprüngliche ACL-Nachricht im ACC kodiert, z. B. in String-Form. Durch diesen Schritt erhält man *payload*, d. h. die zu signierenden Daten. Da *algorithm* mit „RSA_WITH_SHA1" festgelegt ist, wird über diese Daten nun mit dem SHA-1- Algorithmus ein Hash berechnet. Die daraus entstehenden 20 Byte werden mit dem privaten Teil des „AGENT_KEY1" verschlüsselt. Das Ergebnis wird Base64 kodiert und in das Feld *data* eingetragen. In das Feld *key-size* wird die Bytelänge des Modulus von „AGENT_KEY1" eingetragen. Im Anschluss daran werden die Daten vom MTS übertragen. Die eigentliche Sicherungsschicht liegt somit zwischen ACC und MTS.

Auf der Empfängerseite muss die *payload* Signatur vor ihrer Dekodierung verifiziert werden. Dazu entschlüsselt der Empfänger zuerst den Inhalt des Feldes *data*, d. h. die Base64-Daten werden wieder in ein Bytearray umgewandelt, das im Anschluss daran mit dem öffentlichen Teil des „AGENT_KEY1" entschlüsselt wird. Danach berechnet der Empfänger seinen eigenen Hash über *payload* und vergleicht diesen mit den entschlüsselten Daten. Stimmen die Werte überein wird *payload* dekodiert und an den Agenten weitergereicht.

Verschlüsselung

Die Kommunikationssicherung in CARE verfolgt einen hybriden Ansatz, d. h. jede verschlüsselte Nachricht wird mit einem symmetrischen Einmalschlüssel verschlüsselt. Der symmetrische Schlüssel selbst wird RSA-verschlüsselt und der Nachricht im Feld *data* hinzugefügt. Zum Verschlüsseln der *payload* füllt der Agent ein *security-object* mit den gewünschten Werten. Dies kann z. B. wie folgt aussehen:

```
type = "ENCRYPT"
format = "fipa-1"
algorithm = "RSA_CBC3DES"
key-ref = "AGENT_KEY2"
```

Zuerst wird wiederum die ursprüngliche ACL-Nachricht im ACC kodiert, z. B. in String-Form. Durch diesen Schritt erhält man die *payload*, d. h. die zu verschlüsselnden Daten. Da *algorithm* mit dem Wert „RSA_CBC3DES" belegt ist, wird nun, mit Hilfe eines Zufallszahlengenerators, ein 16 Byte langer Einmalschlüssel generiert. *Payload* wird ISO-gepaddet, d. h. es wird ein Byte hex 0x80 an *payload* angefügt und, falls die Länge noch nicht modulo 8 ist, noch mit 0 bis 7 hex 0x00 aufgefüllt. Dies ist notwendig, da Triple DES mit einer Blocklänge von 8 Byte arbeitet. Nach dem Padding wird *payload* mit dem Einmalschlüssel Triple DES CBC verschlüsselt. Der Einmalschlüssel selbst wird mit dem privaten Teil des „AGENT_KEY2" verschlüsselt. Beide Ergebnisse werden Base64 kodiert. Die verschlüsselte ACL-Nachricht verbleibt in *payload.* Der verschlüsselte Einmal-

schlüssel wird in das Feld *data* eingetragen und das Feld *key-size* wird mit der Länge des Einmalschlüssels in Byte befüllt. Im Anschluss daran werden die Daten vom MTS übertragen und der Einmalschlüssel auf der Senderseite verworfen.

Auf der Empfängerseite muss *payload* vor seiner Dekodierung entschlüsselt werden. Dazu entschlüsselt der Empfänger zuerst den Inhalt des Feldes *data*, d. h. die Base64-Daten werden wieder in ein Bytearray gewandelt, das im Anschluss daran mit dem öffentlichen „AGENT_KEY2“ entschlüsselt wird. Danach verwendet der Empfänger *key-size* Byte von *data* als Einmalschlüssel zur Entschlüsselung von *payload*. Im Anschluss daran werden die Paddingdaten entfernt, *payload* dekodiert und an den Agenten weitergereicht.

So wie die ACL-Nachricht eigens kodiert wird, wird auch der *envelope* kodiert. CARE verwendet für die Darstellung des *envelope* XML. Die XML Darstellung des *user-defined* Slots mit zwei *security-objects* sieht wie folgt aus:

```
<user-defined>
<X-security>

<security-object>
<type>SIGN</type>
<format>fipa-1</format>
<security-info>
      <algorithm>RSA_WITH_SHA1</algorithm>
      <key-ref>AGENT_KEY1</key-ref>
      <key-size>144</key-size>
      <data> ... </data>
</security-info>
</security-object>

<security-object>
<type>ENCRYPT</type>
<format>fipa-1</format>
<security-info>
      <algorithm>RSA_CBC3DES</algorithm>
      <key-ref>AGENT_KEY2</key-ref>
      <key-size>16</key-size>
      <data>...</data>
</security-info>
</security-object>

</X-security>
</user-defined>
```

Werden dieselben Daten mehrmals hintereinander verschlüsselt, ist es wichtig, dass diese auf der Empfängerseite in der entgegengesetzten Reihenfolge entschlüsselt werden. Um dies zu garantieren wird in [FIPA03] folgende XML-Repräsentation vorgeschlagen:

```
<security-object order="1">
...
</security-object>
<security-object order="2">
...
</security-object>
```

In der Implementierung der Kommunikationssicherung in CARE wurde auf diese explizite Reihung verzichtet, da der verwendete XML-Parser „EXPAT“ dafür keine Unterstützung liefert. Diese Schwachstelle des XML-Parsers ist bekannt; eine Korrektur des XML-Parser-Codes lag außerhalb der Aufgabenstellung dieser Arbeit. Bei verschachtelten Verschlüsselungen kann es dadurch in der Kommunikation mit fremden Plattformen bzw. Agenten zu Problemen kommen. Die Reihenfolge in CARE ergibt sich aus der C-Struktur SecurityObject, in der ein Pointer jeweils auf das nächst anzuwendende *security-object* verweist. Beim Parsen der XML-Nachricht auf der Empfängerseite wird darauf geachtet, dass die C-Struktur SecurityObject in der Reihenfolge der eintreffenden Werte befüllt wird und so die Reihung erhalten bleibt. Verschachtelte Verschlüsselungen von CARE zu CARE stellen daher kein Problem dar.

In [FIPA03] ist in der Struktur *security-info* auch ein Feld *key* definiert. Dieses Feld wäre für den Transport eines öffentlichen Schlüssels vorgesehen, wenn das Feld *key-ref* nicht befüllt ist, bzw. der benötigte Schlüssel dem Empfänger nicht zur Verfügung steht. Auf das Feld *key* wurde in der aktuellen Implementierung verzichtet. Das gleichzeitige Versenden des öffentlichen Schlüssels mit der verschlüsselten Nachricht würde viele Vorteile des vorgestellten Ansatzes zerstören. Durch die Verwendung einer hybriden Verschlüsselung sind nicht nur die Daten vor den Augen Unbefugter geschützt, d. h. Vertraulichkeit gewährt, gleichzeitig authentifiziert sich auch der Sender beim Empfänger, da nur der Sender Inhaber des privaten Schlüssels *key-ref* sein darf. Wird aber der öffentliche Schlüssel des Senders mitgeschickt, könnte man genauso gut auf eine Verschlüsselung verzichten. In diesem Fall wäre jeder beliebige Empfänger, d. h. auch ein böswilliger Angreifer, der die Kommunikation belauscht, in der Lage den symmetrischen Einmalschlüssel und somit auch die *payload* Daten zu entschlüsseln. Die Signatur hätte ebenso keinen Wert mehr, denn ein Angreifer könnte die Nachricht abfangen, die Daten manipulieren, danach mit seinem eigenen privaten Schlüssel signieren und den dazugehörigen öffentlichen Schlüssel in das Feld *key* eintragen. Der Empfänger der Nachricht hat keine Möglichkeit zu überprüfen, ob der im Feld *key* eingetragene Schlüssel auch wirklich der öffentliche Schlüssel des Absenders ist.

Wie zuvor beschrieben, erfüllt *algorithm* = RSA_CBC3DES bereits zwei der drei zu Beginn des Kapitels definierten Anforderungen, Vertraulichkeit und Authentizität. Die Integrität der Daten wird mit Hilfe von *algorithm* = RSA_WITH_SHA1 sicher gestellt, gleichzeitig wird auch hier wieder der Sender gegenüber dem Empfänger authentifiziert.

Ein Problem, das während der Implementierung auftrat, ist die Verteilung der RSA-Schlüssel. Aktuell sind diese im Agentencode festkodiert enthalten und werden gemeinsam mit dem Programm installiert. Wie bereits im Abschnitt über PKI beschrieben, ist es bei der Wartung eines Agentensystems wichtig, darauf zu achten, dass keine unbefugten Schlüssel in das System eingeschleust werden. So könnte der öffentliche Schlüssel eines Angreifers unter der ID einer vertrauenswürdigen Plattform eingetragen werden. CARE verfügt aber über kein AMS, das bei dieser Aufgabe unterstützen könnte. Auch war eine genaue Zuordnung des Schlüssels zu Agent bzw. Plattform nicht möglich. Die kryptographischen Funktionen kommen zwischen ACC und MTS zur Anwendung. Hier muss auch auf die Schlüssel zugegriffen werden. Da CARE eine Plattform in Form einer C-Bibliothek ist und der Agent aus ressourcensparenden Gründen so klein wie möglich gehalten wird, kommt es hier zu einer Vermischung der Grenzen. Da die Plattform die eigentlichen Funktionen zur

Verfügung stellt, ist eher sie und nicht der Agent als Besitzer der Schlüssel zu betrachten, auch wenn ihre Anwendung (ENRYPT oder SIGN) vom Agenten bestimmt wird.

8. Zusammenfassung und Ausblick

„It is an equal failing to trust everybody,
and to trust nobody."
(Sprichwort)

Einige der in Kapitel 6 vorgestellten Maßnahmen, wie z. B. *Encrypted Functions* sind zwar vielversprechend, aber noch nicht ausgereift genug, um für einen breiteren Einsatz geeignet zu sein. Andere Maßnahmen, wie z. B. *Time-Limited Black Boxes* sind schwer in die existierenden Standards für Agentensysteme zu integrieren. Für die Verbreitung von mobilen Agenten werden aber Standards benötigt, und zwar besonders Sicherheitsstandards, die über das Absichern der bloßen Kommunikation hinausgehen, z. B. muss dabei, wie in [R01c] gefordert und im Abschnitt 3.6 vorgestellt, auch ein statischer Kern zur Absicherung des Agenten definiert werden. Dieser statische Kern, dient dem Agenten als Sicherheitsanker in einer ansonsten potentiell böswilligen Umgebung. Während des Entwurfs der existierenden Standards für Agentensysteme war das Thema „Sicherheit“ kein Schwerpunkt. Ebenso wurden viele der existierenden Plattformen erst nachträglich mit Sicherheitspaketen ausgestattet. Für ein System, das auch einen „sicheren“ Einsatz mobiler Agenten im Internet ermöglicht, sollte dies aber bereits zu Entwurfszeiten eine Voraussetzung sein. SeMoA ist hier eine Ausnahme, da hier schon während des Entwurfs der Plattform besonders auf die Umsetzung der Sicherheitsanforderungen Wert gelegt wurde. Ein nachträgliches Einfügen von „Sicherheit“ in existierende Standards ist schwierig und führt zu weniger sauberen Lösungen und Konflikten mit den bestehenden Definitionen. Wie Abschnitt 3.7.2 zeigt, wäre es notwendig die bereits vorhandenen Standardisierungen einschränkender zu definieren, d. h. Felder mit Längenbegrenzungen zu versehen und Dienste müssten vor ihrem Aufruf eine gegenseitig Authentifizierung verlangen.

Java wurde im Laufe der Jahre, auf Grund seiner Unterstützung für mobilen Code, zu einer Art „de facto“-Standard in der Entwicklung von mobilen Agenten, sowohl für die Realisierung der mobilen Agenten selbst als auch für die Umsetzung der Plattformen. Wie einige der im Kapitel 5 durchgeführten Angriffe zeigen, birgt Java Gefahren in sich. Roth et al. bezeichnen Java in [BR02] als das Beste und gleichzeitig das Schlechteste, das mobilen Agenten zustoßen konnte. Das Beste, da die Entwicklung von mobilen Agentensystemen dadurch einfach wurde, das Schlechteste, da es nahezu

unmöglich ist die Vorteile von Java zu bewahren und gleichzeitig ein genügend sicheres System zu erhalten.

Wie auch Roth in [R01a] feststellt, scheinen die Entwicklungen rund um die Sicherheit mobiler Agenten, i. e. die Entwicklung sicherer Protokolle, um 1998 zum Stillstand zu kommen. Einige Jahre danach geschieht gleiches mit den Standardisierungsansätzen rund um die Sicherheit mobiler Agenten. Mobilen Agenten scheint es an der entsprechenden Lobby zu fehlen. Militärisches Interesse ist kaum vorhanden, das viel zitierte Beispiel bezüglich Reservierung und Kauf eines Flugtickets gibt es in der Realität noch nicht. Am meisten gelangen mobile Agenten zurzeit in abgeschlossenen Systemen zum Einsatz, wie auch das Beispiel PABADIS PROMISE zeigt. Möglich auch, das die vielen offenen Fragen rund um die Realisierung von Sicherheit in mobilen Agentensystemen und die damit verbundenen Kosten bei der Realisierung die Wirtschaft an Client/Server-Anwendungen festhalten lässt. Die rasante Entwicklung am Hardwaremarkt, die Geschwindigkeit betreffend, spielt dabei wohl auch eine Rolle. Auch müssten sich einige Anbieter von Dienstleitungen finden, die bereit sind, ihre Leistungen über Agentenplattformen anzubieten und so zueinander in Konkurrenz zu treten. Preiskampf und wirtschaftlicher Druck würden dabei aber für den einzelnen Anbieter noch schwieriger, da es einem Kunden leichter wird, wesentlich schneller, wesentlich mehr Angebote, als auf dem herkömmlichen Weg möglich ist, einzuholen. Es ist aber denkbar, dass gerade im Bereich mobiler Endgeräte ein derartiger Zusammenschluss erfolgen könnte, da z. B. PDAs (*Portable Digital Assistant*) immer mehr zu Universalgeräten werden und lokale Besonderheiten wie Restaurants aufzeigen oder als Stadtplan dienen. Aufgrund der Tatsache, dass diese mobilen Endgeräte auch abgeschaltet werden bzw. gerade im Ausland die Kosten für länger andauernde Verbindungen zu hoch sind, scheinen Client/Server-Anwendungen hier nicht zielführend zu sein.

Letztendlich gilt für die Umsetzung von „Sicherheit“ in mobilen Agenten, die Kosten-Nutzen-Regel. Das System muss soweit abgesichert sein, dass der Schaden eines möglichen Angriffs kalkulierbar bleibt, bzw. die Kosten für die Realisierung der dazu benötigten Sicherheit auch wirtschaftlich tragbar sind. Kleinere Anwendungen, die lediglich unkritische Informationen sammeln, können mit geringeren Sicherheitsmaßnahmen auskommen; Applikationen, die Käufe tätigen und somit elektronisches Geld mit sich führen, müssen gut abgesichert sein. Einsätze von mobilen Agenten im E-Commerce sind möglich, jedoch nur mit entsprechendem Sicherungs- und auch Protokollierungsaufwand, um im rechtlichen Streitfall entsprechend argumentieren zu können. Der Einsatz qualifizierter elektronischer Signaturen in mobilen Agenten ist rechtlich nicht möglich, da sich die Signatur nicht mehr unter der administrativen Kontrolle ihres Eigentümers befindet, dies hat eine Umkehr der Beweislast zur Folge. Bei Verwendung einer einfachen oder fortgeschrittenen elektronischen Signatur muss in einem Rechtsstreit die sich auf die Signatur beziehende Partei beweisen, dass die Signatur auch wirklich echt ist. Bei Verwendung einer qualifizierten elektronischen Signatur könnte sie bereits von der Echtheit der Signatur ausgehen. Bei Verwendung mobiler Agenten im E-Commerce sollten Maßnahmen gesetzt werden, die das maximal auszugebende Geld beschränken. Da Agenten schlecht Geheimnisse mit sich führen können, ist ein Transport der Kreditkartennummer bei aktuellem Stand der Technik, eher nicht anzuraten. Hier wäre die im Abschnitt 3.2.6 vorgestellte Anwendung von anonymen Chipkarten zu empfehlen. Für hochkritische Anwendungen stellt es ein Problem dar, dass ein mobiler Agent keine Kontrolle über seine Laufzeitumgebung hat und sein einziger „Sicherheitsanker“ sein statischer Code und seine statischen Daten sind.

Ein Patentrezept zum Schutz mobiler Agentensysteme wurde noch nicht gefunden und will auch diese Arbeit nicht liefern, sie soll viel mehr ein Kompendium an Lösungsmöglichkeiten geben. Die vorgestellten Maßnahmen decken jeweils bestimmte Sicherheitsanforderungen ab und müssen für jede Anwendung individuell untereinander kombiniert werden. I. A. kann eine richtig angewandte und konfigurierte Kombination „genug“ Sicherheit für nicht hochkritische Anwendungen liefern. Diese Arbeit kann während des Entwurfs und der Programmierung eines mobilen Agentensystems helfen die Sicherheitsaspekte im Auge zu behalten. Sie zeigt Lücken in bestehenden Standards und Agentensystemen auf und liefert Vorschläge und Entscheidungshilfen zu Sicherungsmaßnahmen, indem sie auf die Vor- und Nachteile der einzelnen Methoden verweist.

Als Schlußwort eignet sich ein Zitat aus [PK00]: „*..., the future of commercial applications of mobile agents is hardly predictable; what is predictable is a promising future for security experts.*“

Anhang A – Definition der FIPA-Strukturen

Dieser Anhang enthält die vollen Definitionen jener FIPA-Strukturen, die im Kapitel 7 benötigt werden.

```
typedef struct SecurityInfo
{     char                  *algorithm;
      char                  *keyRef;
      char                  *keySize;
      char                  *data;
} SecurityInfo;

typedef struct SecurityObject
{     char                  *type;
      char                  *format;
      SecurityInfo          *secInfo;
      struct SecurityObject *nextSecObj;
} SecurityObject;

typedef struct UserDefinedList
{     SecurityObject          *secObj;
      char                    *userDefined;
      char                    *href;
      boolean                 hasNext;
      void                    *next;
      void                    *before;
} UserDefinedList;

// ACLMessage
typedef struct ACLMessage
{     char         *performative;
      AID          *sender;
      AIDList      *receiver;
      AIDList      *replyTo;
      char         *content;
      char         *language;
      char         *encoding;
      char         *ontology;
      char         *protocol;
      char         *conversationID;
      char         *replyWith;
      char         *inReplyTo;
      char         *replyBy;
      char         *time;
} ACLMessage;

// Envelope
typedef struct Envelope
{
      AIDList     *to;
      AID         *from;
      char        *comments;
      char        *ACLRepresentation;
```

```
      char        *payloadLength;
      char        *payloadEncoding;
      char        *date;
      char        *encrypted;
      AIDList     *intendedReceiver;
      Received    *received;
      UserDefinedList   *userDefined;
      void        *currentElement;
} Envelope;
```

Literatur

[AB04] Alfalayleh, Mousa; Brankovic, Ljiljana: *An Overview of Security Issues and Techniques in Mobile Agents*; Proceedings of the Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS); 2004

[B01] Buchmann, Johannes: *Einführung in die Kryptographie*; Springer; 2001

[BEHMW06] Bürkle, Axel; Essendorfer, Barbara; Hertel, Alice; Müller, Wilmuth; Wieser, Martin: *A Test Suite for the Evaluation of Mobile Agent Platform Security*; Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology**;** Seiten 752-756; 2006

[BHMW08] Bürkle, Axel; Hertel, Alice; Müller, Wilmuth; Wieser, Martin: *Evaluating the security of mobile agent platforms*; Autonomous Agents and Multi-Agent Systems; Springer Netherlands; 2008

[BNS05] Bergfelder, Martin; Nitschke, Tanja; Sorge, Christoph: *Signaturen durch elektronische Agenten - Vertragsschluss, Form und Beweis;* Informatik-Spektrum; Springer Berlin/Heidelberg; Juni 2005

[BR02] Binder, Walter; Roth, Volker: *Secure Mobile Agent Systems Using Java: Where are we heading?*; Proceedings of the ACM Symposium on Applied Computing : Universidad Carlos III De Madrid, Madrid, Spain; Seiten 115-119; March 11 - 14, 2002

[BR05] Braun, Peter; Rossak, Wilhelm: *Mobile Agents, Basic Concepts, Mobility Models, & The Tracy Tooklit;* Elsevier. Inc. (USA) und dpunkt.verlag (Deutschland); 2005

[BT06] Bratukhin, Alexsey, Treytl, Albert: *Applicability of RFID and Agent-Based Control for Production Identification in Distributed Production*; ETFA 06. IEEE Conference on Emerging Technologies and Factory Automation; Seiten 1198 – 1205; 2006

[DK06] Diepolder, S.; Krempels, K.H.: *Mobile Agenten*; Praxis der Informationsverarbeitung und Kommunikation; Band 29, Heft 4; K. G. Saur Verlag München; Seiten 203–207; Oktober-Dezember 2006

[EAC99] Edjlali, Guy; Acharya, Anurag; Chaudhary, Vipin: *History-Based Access Control for Mobile Code*; Lecture Notes in Computer Science: Secure Internet Programming; Springer Berlin/Heidelberg; Seiten 413-431; 1999

[FA07] Fragkakis, Michail; Alexandris, Nikolaos: *Comparing the Trust and Security Models of Mobile Agents;* Third International Symposium on Information Assurance and Security; Seiten 363-368; 2007

[FG96] Franklin, Stan; Graesser, Art: *Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents*; Lecture Notes In Computer Science: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages; Springer Verlag London; Seiten 21-35; 1996

[FGS96] Farmer, William M; Guttman, Joshua D.; Swarup, Vipin: *Security for Mobile Agents: Authentication and State Appraisal*; Proceedings of the 4th European Symposium on Research in Computer Security: Computer Security; Springer Verlag London; Seiten 118-130; 1996

[FIPA98] FIPA: *Agent Security Management (Obsolete)*; FIPA 98 Specification, Part 10, Version 1.0; http://www.fipa.org/specs/fipa00020/index.html (abgerufen im September 2008); OC00020A.pdf; Oktober 1998

[FIPA02a] FIPA: *FIPA Abstract Architecture Specification*; http://www.fipa.org/specs/fipa00001/index.html (abgerufen im September 2008); SC00001L.pdf; Dezember 2002

[FIPA02b] FIPA: *FIPA Communicative Act Library Specification*; http://www.fipa.org/specs/fipa00037/index.html (abgerufen im August 2008); C00037J.pdf; Dezember 2002

[FIPA02c] FIPA: *FIPA Agent Management Support For Mobility Specification (Deprecated)*; http://www.fipa.org/specs/fipa00087/index.html (abgerufen im September 2008); DC00087C.pdf; Mai 2002

[FIPA03] FIPA: *FIPA Agent Message Envelope Security Proposal*; http://www.elec.qmul.ac.uk/staffinfo/stefan/fipa-security/documents/Message_security-Spec-2004-03-23.pdf (abgerufen im Oktober 2008); Dezember 2003

[FIPA04] FIPA: *FIPA Agent Management Specification*; http://www.fipa.org/specs/fipa00023/index.html (abgerufen im August 2008); SC00023K.pdf; März 2004

[FS00] Fischmeister, Sebastian: *Building Secure Mobile Agents, The Supervisor-Worker Framework*; Diplomarbeit; Technische Universität Wien, Informationssysteme Institut, Abteilung Verteilte Systeme; 2000

[FVK01] Fischmeister, Sebastian; Vigna, Giovanni; Kemmerer, Richard A.: *Evaluating the Security of Three Java-Based Mobile Agent Systems*; Lecture Notes in Computer Science: Mobile Agents; Springer Berlin/Heidelberg; Seiten 31-41; 2001

[G03] Giansiracusa, Michelangelo: *Mobile Agent. Protection. Mechanisms, and the Trusted Agent Proxy Server (TAPS) Architecture*; Information Security Research Center; Australien; Februar 2003

[GLPR07] Gitter, Rotraud; Lotz, Volkmar; Pinsdorf, Ulrich; Roßnagel, Alexander: *Sicherheit und Rechtsverbindlichkeit mobiler Agenten;* Deutscher Universitätsverlag | GWV Fachverlage GmbH; 2007

[H98] Hohl, Fritz: *Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts*; Lecture Notes in Computer Science: Mobile Agents and Security; Springer Berlin/Heidelberg; Seiten 92-113; 1998

[H01] Hohl, Fritz: *Sicherheit in Mobile-Agentensystemen*; Doktorarbeit; Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Institut für Parallele und Verteilte Systeme, Abteilung Verteilte Systeme; 2001

[ISO7816-4] *ISO 7816-4: Identification Cards – Integrated Circuit(s) Cards, Part 4:Organization, security and commands for interchange*; ISO; 2005

[J06] Josefsson, S.: *RFC4648 - The Base16, Base32, and Base64 Data Encodings*; Network Working Group; http://rfc.net/rfc4648.html (abgerufen im Oktober 2008); Oktober 2006

[J99] Jansen, Wayne A.: *Mobile Agents And Security*; National Institute of Standards and Technology; Canadian Information Technology Security Symposium; Mai 1999

[JK99] Jansen, Wayne; Karygiannis, Tom: *NIST Special Publication 800-19 – Mobile Agent Security;* National Institute of Standards and Technology, Computer Security Division, Gaithersburg, MD 10899; Oktober 1999

[JW98] Jennings, N. R.; Woolridge, M.: *Applications Of Intelligent Agents*; Agent technology: foundations, applications, and markets; Springer-Verlag New York, Inc.; Seiten 3-28; 1998

[K98] Karnik, M. Neeran: *Security in Mobile Agent Systems*; Doktorarbeit; Department of Computer Science and Engineering; Universität von Minnesota; 1998

[KAG98] Karjoth, Günter; Asokan, N.; Gülcü, C.: *Protecting the Computation Results of Free-Roaming Agents*; Lecture Notes In Computer Science: Proceedings of the Second International Workshop on Mobile Agents; Springer Verlag/London UK; Seiten 195-207; 1998

[LAH04] Lee, Hyungjick; Alves-Foss, Jim; Harrison, Scott: *The Use of Encrypted Functions for Mobile Agent Security*; Proceedings of the 37th Annual Hawaii International Conference on System Sciences; 2004

[M98] Mattern, Friedemann: *Mobile Agenten*; it+ti - Informationstechnik und Technische Informatik, 4/98; Seiten 12-17; 1998

[MBB99] Milojicic, Dejan; Breugst, Markus; Busse, Ingo; Campbell, John; Covaci, Stefan; Friedman, Barry; Kosaka, Kazuya; Lange, Danny; Ono, Kouichi; Oshima, Mitsuru; Tham, Cynthia;

Virdhagriswaran, Sankar; White, Jim: *MASIF – The OMG Mobile Agent System Interoperability Facility*; Mobility: processes, computers, and agents; ACM Press/Addison-Wesley Publishing Co; Seiten 628–641; 1999

[MOV97] Menezes, Alfred J.; van Oorschot, Paul C.; Vanstone, Scott A.: *Handbook of Applied Cryptography;* CRC Press LLC; 1997

[NIST99] *Federal Information Processing Standards Publication 46-3: DATA ENCRYPTION STANDARD*; U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology; http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf (abgerufen im Oktober 2008); Oktober 1999

[NL98] Necula, George C.; Lee, Peter: *Safe, Untrusted Agents Using Proof-Carrying Code*; Herausgeber: Vigna, Giovanni: *Mobile Agents and Security*; Springer Berlin/Heidelberg; Seiten 61-91; 1998

[NPR06] Nitschke, Lukas; Paprzycki, Marcin; Ren, Michal: *Mobile Agent Security*; J. Thomas, M. Essaidi (Hrsg.): Information Assurance and Computer Security; IOS Press, Amsterdam; Seiten 102-123; 2006

[O96] Ordille, Joann J.: *When Agents Roam, Who Can You Trust?*; Proceedings of the First conference on Emerging Technologies and Applications in Communications; Portland, Oregon; Mai 1996

[OC07] Odubiyi, Jidé B.; Choudhary, Abdur Rahim: *Building Security into an IEEE FIPA Compliant Multiagent System*; Information Assurance and Security Workshop; Seiten 49-55; Juni 2007

[OMG97] OMG: *Mobile Agent System Interoperability Facilities Specification;* http://www.omg.org/cgi-bin/doc?orbos/97-10-05 (abgerufen im September 2008); GMD FO KUS, International Business Machines Corporation; November 1997

[OMG00] OMG: *Mobile Agent Facility*; http://www.omg.org/cgi-bin/doc?formal/2000-01-02 (abgerufen im September 2008); GMD FOKUS, IBM; Jänner 2000

[PK00] Posegga, Joachim; Karjoth, Günter: *Mobile Agents and Telcos' Nightmares*; Annales des Telecomunication, special issue on communications security; 2000

[PPW02] Pichler, J.; Plösch, R.; Weinreich, R.: *MASIF und FIPA: Standards für Agenten, Übersicht und Anwendung*; Informatik Spektrum, Band 25; Springer Verlag; Seiten 91-100; April 2002

[R98] Roth, Volker: *Secure Recording of Itineraries through Cooperating Agents*; In Proc. 4th ECOOP Workshop on Mobile Object Systems: Secure Internet Mobile Computations; Brüssel, Belgien; Seiten 147-154; Juli 1998

[R00] Roth, Volker: *Mobile Software Agenten und Sicherheit – eine Haßliebe?*; http://www.volkerroth.com/download/Roth2000b.pdf (abgerufen im September 2008); Fraunhofer Institut für Graphische Datenverarbeitung; 2000

[R01a] Roth, Volker: *Programming Satan's Agent;* 1st International Workshop on Secure Mobile Multi-Agent Systems; Montreal, Canada; 2001

[R01b] Roth, Volker: *On the Robustness of some Cryptographic Protocols for Mobile Agent Protection;* Lecture Notes In Computer Science: Proceedings of the 5th International Conference on Mobile Agents; Springer Verlag/London UK; Seiten 1-14; 2001

[R01c] Roth, Volker: *Über die Bedeutung eines statischen Kernes für die Sicherheit Mobiler Software-Agenten*; Kommunikationssicherheit im Zeichen des Internet, DuD Fachbeiträge, Seiten 227-234; Vieweg-Verlag; März 2001

[RC01] Roth, Volker, Conan, V.: *Encrypting Java Archives and its application to mobile agent security*; Lecture Notes in Computer Science: Agent Mediated Electronic Commerce; Springer Berlin/Heidelberg; Seiten 229-239; 2001

[RG98] Rubin, Aviel D.; Geer, Daniel E.: *Mobile Code Security*; IEEE Internet Computing; Volume 2; Seiten 30-34; November 1998

[RJ01] Roth, Volker; Jalali-Sohi, Mehrdad: *Concepts and Architecture of a Security-Centric Mobile Agent Server*; Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems; IEEE Computer Society Press; Seiten 435-442; 2001

[RJP01] Roth, Volker; Jalali, Mehrdad; Pinsdorf, Ulrich: *Secure Mobile Agents: SeMoA*; COMPUTER GRAPHICS topics (Unternehmensmagazin des INI-GraphicsNet); Heft 1; 2001

[RS98] Riordan, James; Schneier, Bruce: *Environmental Key Generation Towards Clueless Agents*; Lecture Notes in Computer Science: Mobile Agents and Security; Springer Berlin/Heidelberg; Seiten 15-24; 1998

[S96] Schneier, Bruce: *Applied Cryptography, Protocols, Algorithms and Source Code in C*; 2. Auflage; John Wiley & Sons, Inc.; 1996

[S97] Schneider, Fred B.: *Towards Fault-Tolerant and Secure Agentry*; Lecture Notes In Computer Science: Proceedings of the 11th International Workshop on Distributed Algorithms; Springer Verlag London; Seiten 1-14; 1997

[S07] Spenger, Werner Josef: *Laufzeitumgebung für Software-Agenten auf RFID-Basis*; Diplomarbeit; TU Wien, Fakultät für Elektrotechnik und Informationstechnik, Institut für Computertechnik; Dezember 2007

[ST97] Sander, Tomas; Tschudin, Christian F.: *Towards Mobile Cryptography*; International Computer Science Institute (ICSI); Technical Report TR-97-049; November 1997

[ST98] Sander, Tomas; Tschudin, Christian F.: *On Software Protection Via Function Hiding*; Lecture Notes in Computer Science: Information Hiding; Springer Berlin/Heidelberg; Seiten 111-123; 1998

[TSR08] Treytl, Albert; Spenger, Werner; Riaz, Bilal: *A secure agent platform for active RFID*; ETFA 2008 IEEE International Conference on Emerging Technologies and Factory Automation; Seiten 492 – 495, 2008

[V98] Vigna, Giovanni: *Mobile Code Technologies, Paradigms, and Applications*; Doktorarbeit; Politecnico di Milano, Italien; Februar 1998

[VB01] Villazòn, Alex; Binder, Walter: *Portable Resource Reification in Java-based Mobile Agent Systems*; Lecture Notes in Computer Science: Mobile Agents; Springer Berlin/Heidelberg; Seiten 213-228; 2001

[YY97] Young, Adam; Yung, Moti: *Sliding Encryption: A Cryptographic Tool For Mobile Agents*; Lecture Notes in Computer Science: Fast Software Encryption; Springer Berlin/Heidelberg ; Seiten 230-241; 1997

[Y99] Yee, Bennet S: *A Sanctuary for Mobile Agents*; Lecture Notes in Computer Science: Secure Internet Programming; Springer Berlin/Heidelberg; Seiten 261-273; 1999

Internetreferenzen

[AB-WIKI] http://de.wikipedia.org/wiki/Anscheinsbeweis; abgerufen im Oktober 2008

[AGLETS] http://www.trl.ibm.com/aglets/; abgerufen im September 2008

[AZ-WIKI] http://de.wikipedia.org/wiki/Attributzertifikat; abgerufen im Juli 2008

[CORBA-WIKI] http://de.wikipedia.org/wiki/CORBA; abgerufen im September 2008

[FIPA] http://www.fipa.org/; abgerufen im August 2008

[H-WIKI] http://de.wikipedia.org/wiki/Homomorphismus; abgerufen im September 2008

[OMG-INTRO] http://www.omg.org/gettingstarted/specintro.htm#OMA; abgerufen im September 2008

[OTP-WIKI] http://de.wikipedia.org/wiki/One-Time-Pad; abgerufen im Juli 2008

[IKV] http://www.presseportal.de/pm/22506/159366/ikv_gmbh; abgerufen im September 2008

[REFLECT] http://www.dpunkt.de/java/Die_Sprache_Java/Objektorientierte_Programmierung_mit_Java/70.html; abgerufen im September 2008

[RFID-WIKI] http://de.wikipedia.org/wiki/RFID; abgerufen im Oktober 2008

[RP-WIKI] http://de.wikipedia.org/wiki/Replay-Angriff; abgerufen im Juli 2008

[SEMOA] http://semoa.sourceforge.net/about/details.html; abgerufen im Juli 2008

[SIG-AT] http://www.signatur.rtr.at/de/legal/index.html; Rundfunk und Telekom Regulierungs-GmbH; abgerufen im Juli 2008

[SIG-DE] http://www.bundesnetzagentur.de/enid/Sachgebiete/Qualifizierte_elektronische_Signatur_gz.html; Die Bundesnetzagentur – Qualifizierte elektronische Signatur; abgerufen im Juli 2008

[SIG-WIKI] http://de.wikipedia.org/wiki/Elektronische_Signatur; abgerufen im Oktober 2008

[SIG-QES] http://digitale-signatur.edulab.de/signaturgesetz.html; abgerufen im Oktober 2008

[WB-WIKI] http://de.wikipedia.org/wiki/W%C3%B6rterbuch-Angriff (Wörterbuchangriff); abgerufen im Juli 2008