Entwicklung eines sicheren, erweiterten E-Mail-Systems

Thomas Hildmann, Matr.Nr. 130931

Fachbereich Informatik
Institut für Kommunikations- und Softwaretechnik
Technische Universität Berlin
Prof. Dr.-Ing. Stefan Jähnichen

Eidesstattliche Erklärung

Diese Diplomarbeit "Entwicklung eines sicheren, erweiterten E-Mail-Systems" sowie die beiliegenden Quelltexte und Dokumentationen auf der CD unter dem Verzeichnis /sme/ wurden von mir selbständig und eigenhändig angefertigt.

Berlin, den 13. November 1998

_____

Thomas Hildamnn

Inhalt

KAPITEL 1

Einleitung

Obwohl in den Medien das WWW als der wichtigste Internetdienst beschrieben wird und für viele Menschen mittlerweile das WWW mit dem Internet synonym verstanden wird, ist in der Geschäftswelt und im Forschungssektor die E-Mail ein wichtiges Hilfsmittel. Der E-Mail Dienst wird hier als schnelle und bequeme Alternative zum Faxgerät, zur internen Kommunikation aber auch in Form von Mailinglisten zur Verteilung von Informationen, Informationsbeschaffung und Weiterbildung verwendet. Auch das Versenden von Dateien über E-Mail z.B. zur Weiterbearbeitung oder als Baustein einer bestimmten Arbeit ist eine beliebte Anwendung. Statt z.B. Faxe zu verschicken, die dann wieder neu eingetippt werden müssen**,** können die Informationen sofort weiterverarbeitet werden. Auch beinhalten moderne E-Mail-Programme Möglichkeiten zur Organisation von E-Mails nach bestimmten Kriterien oder zumindest die Organisation in Ordnern, in denen Briefe nach Themen oder Korrespondenzpartnern sortiert werden können. Es kann in bestehenden Briefen dann nach Stichworten gesucht werden, erhaltene Informationen können ggf. ohne großen Aufwand weitergeleitet werden. Dies ist nur ein kleiner Ausschnitt aus den Möglichkeiten, die aus der Nutzung von E-Mails entstehen.

Leider wird von immer mehr Firmen die E-Mail auch als ein wirksames Werbemedium verstanden. Eine Tatsache, die dem gesamten Medium E-Mail in meinen Augen großen Schaden zufügt. Wirksame Filtermethoden müssen eingesetzt werden, um zu verhindern, daß der E-Mail-Benutzer seine kostbare Zeit damit verschwenden muß, Werbesendungen von wichtiger Post zu trennen. Dies ist ein aufwendiger und fehleranfälliger Prozeß. Leicht gehen hier auch wichtige Informationen verloren. Auf diese Problematik möchte ich in meiner Arbeit jedoch nicht genauer eingehen. Nur möchte ich meinen Apell an alle Leser richten, solches Verhalten zu boykottieren und auf keine per E-Mail verschickte und nicht direkt angeforderte Werbung zu reagieren. Ganz im Gegenteil sollten seriöse Firmen solchen “schwarzen Schafen” vorgezogen werden.

E-Mails sind ein mächtiges Werkzeug, das richtig eingesetzt, anderen Medien gegenüber fantastische Vorteile bietet. In verschiedenen Bereichen gestaltet sich die Einführung von E-Mail Systemen jedoch sehr schwer, da die Anforderungen an eine Kommunikation von der klassischen Form des E-Mail Dienstes nicht erfüllt werden. In dieser Arbeit werde ich am Beispiel der Verwaltung der Technischen Universität Berlin versuchen, die Möglichkeiten eines erweiterten E-Mailsystems zu erarbeiten.

Diese Diplomarbeit entstand im Rahmen des EU-Projektes E2S (End-to-End-Security over the Internet). Ziel des Projekts war das zur Verfügungstellen von sicheren Internetdiensten mit zwei Kommunikationspartnern. Dabei ging das Projekt nicht vornehmlich darauf ein, wie eine Absicherung aus kryptographischer Sicht realisierbar ist. Ziel war es vielmehr, bestehende Technologien so zu kombinieren, daß sie in einem vorgegebenen Kontext bequem und benutzerfreundlich eingesetzt werden können.

In dieser Arbeit werde ich zwei Ebenen von Benutzerfreundlichkeit (in arbeitswissenschaftlicher Literatur wird heute gern von Benutzungsfreundlichkeit gesprochen) betrachten:

1. Der Aufwand, den Endbenutzer als E-Mail Sender und Empfänger betreiben müssen, um sicher E-Mails übertragen zu können.

2. Der Verwaltungsaufwand eines Systemadministrators. Ein erweitertes E-Mail System hätte wohl keine Chance, ernsthaft eingesetzt zu werden, wenn der Verwaltungsaufwand lediglich vom Endbenutzer zum Systemverwalter verschoben würde.

Aber es soll in dieser Arbeit nicht ausschließlich um Benutzungsfreundlichkeit gehen. Es leuchtet ein, daß für den Einsatz in einem Verwaltungsaparat mit personaltechnischen und finanziellen Verwaltungsaufgaben Sicherheit aus mehreren Sichtweisen gefordert wird. Dazu gehört zum einen die Datenkanalsicherung, die ausschließen soll, daß z.B. personenbezogene Daten angegriffen werden können und zum anderen die Ausfallsicherheit des Systems. Es muß möglichst sichergestellt sein, daß jede Mail von praktisch jedem beliebigen Mailclient korrekt von der Software angenommen, verarbeitet und weitergeleitet wird. Ein zentraler Punkt ist folglich die Korrektheit der Software. Ich versuche eine möglichst hohe Verläßlichkeit durch die Auswahl meiner Softwareentwicklungsmethode und dem Einsatz der darunterliegenden Software zu gewährleisten. Von der Hardware versuche ich dabei weitgehend unabhängig zu bleiben, um einen breiten Einsatz möglich zu machen.

Die Verarbeitungsgeschwindigkeit des Systems wird in dieser Arbeit als unproblematisch eingestuft. Das System muß jedoch eine hohe Anzahl von Anfragen abarbeiten können. Beim E-Mail Dienst handelt es sich um einen asynchronen Dienst, bei dem Verzögerungen in der Abarbeitung durchaus in Kauf genommen werden können. Die Geschwindigkeit des erweiterten E-Mail Systems ist vor allem von zwei externen Prozessen abhängig; dem rollenbasierten Zugriffssystem und den externen Ver- und Entschlüsselungssystemen.

1.1 Motivation

Bei dieser Arbeit wird es darum gehen, den E-Mail Dienst so zu erweitern, daß er in einer Verwaltung, wie der der TU Berlin einsetzbar wäre. Die Vorgehensweise ist also die, die Schwächen der bisherigen E-Mail Systeme aufzudecken, zu beseitigen und dabei aber keine Einschränkungen zu erzeugen, die aus Sicht der Verwaltung einen Rückschritt darstellen würde.

Den ersten Entwicklungsschritt stellt die Spezifikation des ARPA-Message-Formats und die Spezifikation des SMTP-Protokolls zum Austausch von E-Mails dar. Dieses wurde durch das ESMTP- und POP3-Protokoll komplettiert. ESMTP beinhaltet lediglich eine Erweiterung des Befehlssatzes bei der Kommunikation von Mailservern untereinander. POP3 bietet die Möglichkeit, von einer entfernten Arbeitsstation E-Mails von einem Server abzufragen. Die große Verbreitung brachte jedoch in den Anfangzeiten das UUCP-Protokoll (Unix-to-Unix-Copy), das es ermöglichte, z.B. auch über Modem E-Mails auszutauschen. Zur Zeit der Großrechner, Terminals und später auch zu Zeiten der ersten nicht sehr leistungsfähigen Homecomputer war die Verwendung von Mailbox-Systemen (oder auch BBS, Bulletin Board Systems genannt) ein großer Fortschritt. Ein Mailboxrechner tauschte in regelmäßigen Abständen z.B. über das UUCP Mails aus, die dann von zu Hause aus über Modem und Terminalprogramm abgefragt werden konnten.

Der nächste Schritt war die Erweiterung der E-Mail mit Multimedia Funktionen. Das Transportieren binärer, typisierter Dateien wurde mit Hilfe des MIME Standards verwirklicht. Dieser Schritt wurde durch den Einsatz multimediafähiger Arbeitsplätze ermöglicht. Heute kann jeder PC eine Internetverbindung über Modem oder ISDN herstellen und mit einem MIME-fähigen Mailprogramm abfragen oder erstellen. Dank ausgereifter Benutzeroberflächen ist keine besondere Qualifikation mehr nötig, um mit E-Mails arbeiten zu können. Heute ist eine E-Mail fast überall auf der Welt innerhalb weniger Minuten ausgeliefert und kann gelesen werden. In manchen Forschungsbereichen wird fast schon

selbstverständlich die Tatsache ausgenutzt, daß Institute in anderen Ländern durch die Zeitverschiebung genau zu dem Zeitpunkt Informationen beschaffen können, wenn im eigenen Land geschlafen wird. Am nächsten Morgen ist die Antwort per E-Mail angekommen.

Die neuesten Entwicklungen gehen in die Richtung der sicheren E-Mail. Hierzu hat sich das Programm PGP als Standard fast durchgesetzt, auch wenn der Schritt zum PGP 5.x einige Probleme mit sich bringt. Hier ist als störender Faktor für die Durchsetzung vor allem das Ausfuhrverbot für “starke” kryptographische Algorithmen aus den USA zu erwähnen.

Es kann folgender aktueller Stand zusammengefaßt werden:

- Die Nutzung des E-Mail-Dienstes ist weit verbreitet. Eine Infrastruktur steht über das Internet zur Verfügung.
- E-Mail Programme (Clients) sind auf praktisch allen Rechnerplattformen verfügbar und unterstützen Funktionen bis hin zu MIME Erweiterungen.
- Sichere E-Mail wird von verschiedenen Clients mit unterschiedlichen Standards angeboten.

Ich wollte in meiner Arbeit auf keine fertige Lösung für sichere E-Mails zurückgreifen, da sie alle gewisse Nachteile mit sich bringen. Dazu gehört die Tatsache, daß das einzige auf fast allen Plattformen verfügbare PGP sehr komplex in seiner Bedienung ist. Selbst die neuere PGP 5.x Version, die sich nicht unbedingt abwärtskompatibel zu seinen Vorgängern zeigt [Cam98], ist zu schwierig für den Einsatz in der Verwaltung. Hierbei ist vor allem die Einschränkung zu nennen, daß die Schlüsselverwaltung vom Benutzer bedient werden muß, auch wenn es in der Version 5.x die Schnittstelle zu einem Keyserver gibt.

Die sicheren E-Mails, die von den wohl am weitesten verbreiteten Mailclients (Netscape Communicator und Microsoft Outlook) zur Verfügung gestellt werden, binden die Benutzer an die externen Schlüsselverwalter, z.B. die Frima Verisign. Auch hier ist die Benutzung noch nicht so einfach, wie das für die Verwaltung gefordert werden sollte.

KAPITEL 2

Techniken und Protokollelemente

2.1 Kryptographie

Die Kryptographie ist eine Wissenschaft mit Tradition. Die Aktivitäten von Kryptographen reichen bis ins Altertum zurück. Nur die Werkzeuge haben sich um Laufe der Zeit geändert. Das Ziel ist jedoch immer das gleiche gewesen: Die Absicherung von Nachrichten. Also die Tatsache, daß eine Nachricht von Person A an Person B gelangt, ohne daß eine unautorisierte Person C an den Inhalt der Nachricht gelangen kann. Man kann zwei Ansätze unterscheiden, wie ein in unserem Sinne abgesicherter Nachrichtenaustausch stattfinden kann. Der eine Ansatz beschäftigt sich mit der Idee, daß Person C vom Austausch der Nachricht keine Kenntnis bekommt. Ein Beispiel hierfür ist das über die Grenze geschmuggelte Magazin, das eine Nachricht im Kreuzworträtsel beinhaltet. Hier geht es darum, die Nachricht zu tarnen. Heute sind ganz andere Formen möglich. So befassen sich Wissenschaftler (und sicherlich nicht nur die) mit dem Verstecken von Informationen in den Kanten von Bildern einer Bewegtbildübertragung oder das Verstecken von Daten in einer Audioübertragung. Der zweite Ansatz ist der kryptographische Ansatz, der nicht die Nachricht an sich versteckt, sondern nur deren Inhalt. Dieses Verbergen der Information wird durch Verschlüsselung erreicht.

Für die Informatik stellt sich das Problem der Verschlüsselung prinzipiell sehr einfach dar. Eine zentrale Frage in der Informatik war seit jeher die Kodierung von Daten; die Frage: “Wie stelle ich einen Informationssachverhalt dar?” Die Problematik besteht nun darin, die Informationen so umzukodieren, daß ein Gewinnen der Ursprungsinformationen nicht ohne eine bestimmte andere Information möglich ist.

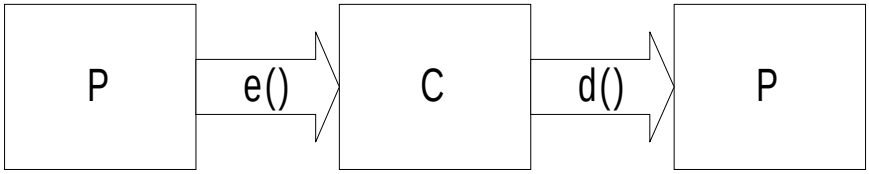
2.2 Verschlüsselung

2.2.1 Prinzip der Verschlüsselung

Das Verfahren, eine Nachricht unverständlich zu machen, um ihren Inhalt zu verbergen, heißt Verschlüsselung. Der Klartext (engl.: plain text) wird in einen Chiffretext (engl.: cipher text) umgewandelt und dann bei der Entschlüsselung wieder in Klartext zurück gewandelt.

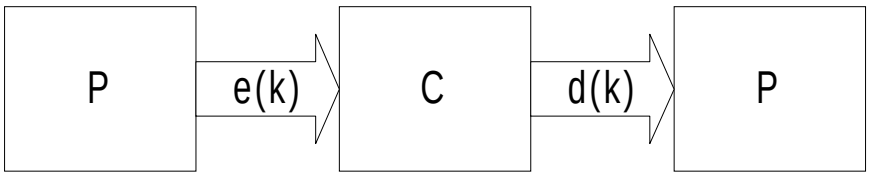
Grundlage einer Verschlüsselung ist eine Funktion e, welche beim Sender einer Nachricht aus einem Klartext P einen chiffrierten Text C erzeugt sowie die inverse Funktion d, welche diese Umwandlung beim Empfänger wieder rückgängig machen kann.

ABBILDUNG 1. Ver- und Entschlüsselung mittels Funktionen e() und d()



Da die Geheimhaltung der eigentlichen Verschlüsselungsfunktionen im allgemeinen nicht auf die Dauer zu gewährleisten ist, werden diese mit geheimen Schlüsseln (k) parameterisiert, welche sie in ihrer Wirkung beeinflussen.

ABBILDUNG 2. Ver- und Entschlüsselung mittels parameterisierter Funktionen



Das Prinzip läßt also zu, daß die Ver- und Entschlüsselungsfunktion allgemein bekannt ist, jedoch das Wissen um diese Funktion einem Angreifer nicht ausreicht, um an den Klartext zu gelangen. Nur mit Hilfe des Schlüssels kann der Klartext wieder hergestellt werden.

Die Art der Information spielt für die Computer-Kryptographie eine sekundäre Rolle. Sollen anstelle von Texten, Bilder, Videos, Tondateien oder andere Daten übertragen werden, ist dies aus Sicht des Algorithmus egal. Die kryptographischen Verfahren, von denen wir hier sprechen, fassen eine beliebige Bitfolge als Klartext auf und können diese völlig transparent in den Chiffretext (wieder eine andere Bitfolge) und wieder zurückwandeln, ohne daß die Information in irgendeiner Weise beeinflußt wird. Im Grunde läßt sich also alles verschlüsseln, was digital erfaßt werden kann. Grenzen sind hier nur durch die Rechenleistung der zur Ver- und Entschlüsselung benutzten Maschinen, der Speichermedien (sowohl der Schlüssel, der Klartext als auch der Chiffretext müssen irgendwo gespeichert werden, auch wenn z.B. ein gleichzeitiges Löschen und Erzeugen, also eine Art Umwandeln denkbar ist, so daß hier Speicherplatz gespart werden kann) und natürlich der Übertragungsmedien (und deren Bandbreite) gesetzt. Somit wird die verschlüsselte Übertragung eines digitalisierten Films evtl. schon zum Problem, da hier zum Beispiel sehr große Mengen an Daten auftreten.

Sehr viele Technologien arbeiten im Internet mit Verschlüsselung. Einem möglichen Angreifer soll der Zugriff auf sensible Informationen so schwer wie möglich gemacht werden. Verschlüsselte Informationen können ohne Kenntnis des für die Dekodierung benötigten Schlüssels nur unter großem Rechenaufwand dechiffriert werden.

Das Problem ist bei der Nutzung des Internets deshalb so groß, weil das Internet nie als abhörsicheres Medium geplant war. Vielmehr wurde hier eine Technik verwendet, die vorhanden war, jedoch den Anforderungen an vielen Stellen gar nicht gewachsen ist. Somit müssen Schutzmechanismen jetzt auf Applikationsebene eingebaut werden. Die neueren Spezifikationen sehen jedoch auch schon Sicherungen auf unteren Netzschichten vor, was die Sicherung auf höheren Ebenen jedoch nicht überflüssig macht.

2.2.2 Authentifizierung, Integrität und Verbindlichkeit

Neben der Geheimhaltung soll die Kryptographie noch andere Anforderungen erfüllen:

Authentifizierung

Der Empfänger sollte jederzeit die Herkunft einer Nachricht ermitteln können. Ein Angreifer sollte sich nicht als eine andere Person ausgeben können. Es soll ein Mechanismus, wie eine Unterschrift oder ein Siegel nachgebildet werden.

Integrität

Der Empfänger sollte überprüfen können, ob die Nachricht bei der Übermittlung verändert wurde; ein Angreifer sollte die echte nicht durch eine gefälschte Nachricht ersetzen können.

Verbindlichkeit (Nicht-Abstreitbarkeit)

Ein Sender sollte später nicht leugnen können, daß er eine Nachricht in dieser Form gesendet hat.

2.2.3 Verschlüsselungsverfahren

Es lassen sich zwei Ansätze für die Verschlüsselung und Dekodierung von Informationen unterscheiden, die symmetrischen und asymmetrischen Verfahren.

Symmetrische Verschlüsselungsverfahren

Bei den symmetrischen Verfahren muß der Schlüssel dem Sender und dem Empfänger gemeinsam bekannt sein. Er stellt also ein gemeinsames Geheimnis (engl.: common secret) der beiden Parteien dar. Das wirft das Problem der Schlüsselverteilung auf. Wie können sich die beiden Parteien im Vorfeld einer Kommunikationsbeziehung über einen gemeinsamen Schlüssel einigen, ohne daß dieses von einem Angreifer abgehört bzw. manipuliert werden kann. Die Frage der Schlüsselverteilung taucht im folgenden immer wieder auf und stellt ein Problem dar, das beim heutigem Stand der Forschung bei der praktischen Umsetzung immer wieder Einschränkungen zur Folge hat.

Jede Person, die Zugang zum gemeinsamen Schlüssel eines symmetrischen Verschlüsselungsverfahren hat, kann die Informationen entschlüsseln. Besonders schwierig wird hier die Schlüsselvergabe, wenn einer ganzen Gruppe ein Schlüssel bekannt ist, sich die Gruppe dann aber in irgendeiner Form ändert, also Personen ausscheiden und andere hinzukommen. Dann muß für alle neuen Gruppenmitglieder wieder ein neuer Schlüssel ausgehandelt werden. Je größer die Gruppe ist, desto größer wird auch die Gefahr, daß der Schlüssel einem Angreifer bekannt wird.

Ein Beispiel - Algorithmus

1917 entwickelte der Ingenieur Gilbert S. Vernam das symmetrische Verschlüsselungsverfahren, das auf der XOR - Funktion beruht. Dazu wird die Nachricht in einen Bitstrom zerlegt (t). Dann wird eine zufällige Bitfolge erstellt, die genau die Länge der Nachricht hat (k). Über beide Bitfolgen wird ein XOR berechnet. Das Ergebnis ist der Chiffretext (c). Es ist mathematisch beweisbar, daß dieses Verfahren sicher ist, da der Originaltext (t) nicht wieder hergestellt werden kann, ohne den Schlüssel (k)

zu besitzen. Anhand des Bitstroms (t) läßt sich anders herum auch der Schlüssel (k) berechnen. Voraussetzung, damit dieses Verfahren funktionieren kann, ist ein Protokoll, das unabhängig die beiden Teile (t) und (k) zum Empfänger transportieren kann, ohne daß es einem Angreifer möglich ist, beide Teile in die Hand zu bekommen.

TABELLE 1. Beispielverschlüsselung mit XOR - Algorithmus

Bezeichnung	Kürzel	Bitmuster
Klartext	t	0011
Schlüssel	k	0101
Chiffretext	c	0110

In der Praxis arbeitet man deshalb in der Regel so, daß ein sehr viel kürzerer Schlüssel (ein Paßwort oder eine Zahlenkombination) auf einem anderen Kommunikationsweg übertragen wird und die Ver- und Entschlüsselung durch wiederholtes Anwenden der Schlüsselkombination bewältigt wird. Der Schlüssel wird also so lange immer wieder aneinandergehängt, bis er die Länge des Klartextes hat. So benutzte Schlüssel können jedoch mit kryptoanalytischen Verfahren (z.B. Wahrscheinlichkeiten von Buchstaben in einer bestimmten Sprache) leicht ermittelt werden. So benutzte XOR-Verfahren gelten in Fachkreisen als Spielzeug. Wird das Verfahren jedoch, wie von Vernam angedacht verwendet, ist es wirklich nur mit untragbarem Aufwand zu knacken [Hag94].

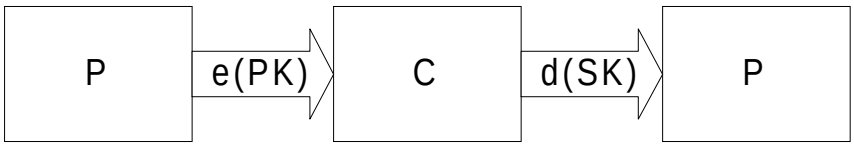
Andere symmetrische Verfahren können auch mit kleineren Schlüsseln eine hohe Sicherheit bieten, sie sind jedoch sehr viel schwieriger zu beschreiben und zu verstehen.

Asymmetrische Verschlüsselungsverfahren

Bei asymmetrischen Verschlüsselungsverfahren wird die zu übertragende Nachricht mit je einem anderen Schlüssel ver- und entschlüsselt. Es existieren also jeweils zwei Schlüssel:

- ein öffentlicher Schlüssel (engl.: public key, Abk.: PK), mit dem ein Sender einer Nachricht die zu sendende Nachricht kodiert.
- ein geheimer Schlüssel auch privater Schlüssel (engl.: secret key, private key, Abk.: SK), mit dem der Empfänger die Nachricht wieder entschlüsseln kann.

ABBILDUNG 3. Verschlüsselung mit einem Parameter, der von Entschlüsselung verschieden ist



In der Praxis werden die Public-Key-Algorithmen oft nur zur Chiffrierung von symmetrischen Schlüsseln verwendet und nicht zum Verschlüsseln von Nachrichten. Dafür gibt es einen einfachen, praktischen Grund:

Im Vergleich zu symmetrischen Verfahren sind auch die besten asymmetrischen Algorithmen wegen ihrer relativ komplexen mathematischen Konzepte äußerst ineffizient. Symmetrische Verfahren sind auf heutigen Rechnern bis zu 1000 mal schneller als asymmetrische Verfahren.

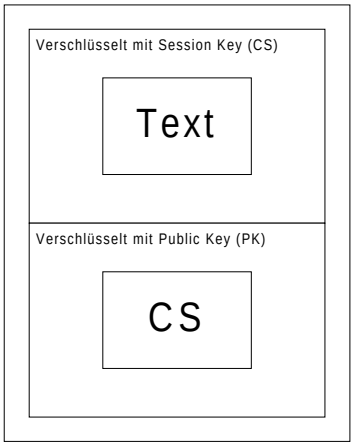
Durch eine ständige Weiterentwicklung werden in den nächsten 15 Jahren auch die asymmetrischen Verfahren so schnell wie heute die symmetrischen sein, allerdings sind bis dahin auch die allgemeinen Anforderungen an Bandbreite und Geschwindigkeit gestiegen [Schn96].

Eine Kombination der beiden Verfahren liefert uns sowohl Sicherheit als auch Schnelligkeit. Man spricht dann von hybriden Verschlüsselungsverfahren.

Es gibt zwei Ansätze für die Kommunikation mit hybriden Verschlüsselungsverfahren:

- Zu Beginn der Kommunikationsbeziehung wird der symmetrische Schlüssel für die Sitzung festgelegt und mit dem öffentlichen Schlüssel des Kommunikationspartners kodiert übertragen. Damit ist der Austausch des Schlüssels gesichert. Die auszutauschenden Informationen müssen dann lediglich mit dem Session Key kodiert werden (symmetrisches Verfahren).
- Die öffentlichen Schlüssel werden auch beim zweiten Ansatz eingesetzt, um den privaten Schlüssel zu kodieren. Dieser wird jedoch mit der Information zusammen übertragen. Während mit dem öffentlichen Schlüssel nur der relativ kurze symmetrische Schlüssel für die Kommunikation kodiert wird, ist der Rest der Informationen bereits mit dem privaten Schlüssel kodiert. Ein solches Protokoll wird als digitaler Umschlag bezeichnet.

ABBILDUNG 4. Darstellung eines digitalen Umschlags (hybride Verschlüsselung)



Der Unterschied zwischen den beiden Ansätzen besteht nur in der Lebensdauer des symmetrischen Schlüssels. Im ersten Fall besteht der symmetrische Schlüssel über die gesamte Lebensdauer der Kommunikation. Im zweiten Fall wird für jede Übertragungseinheit ein neuer Schlüssel festgelegt und mit der Information gemeinsam übertragen. Wenn man in diesem Kontext von symmetrischen Schlüsseln spricht, wird oft der Begriff Session Key verwendet, und man meint damit einen temporär ausgehandelten symmetrischen Schlüssel.

Ich werde in einem späteren Kapitel zeigen, wie man die Tatsache ausnutzen kann, daß sowohl der PGP- als auch der PEM-Standard jeweils hybride Verfahren einsetzen, die darauf basieren, daß der Text der E-Mail mit einem symmetrischen Verfahren verschlüsselt wird und der symmetrische Schlüssel dann mit einem PK-Verfahren verschlüsselt und an den verschlüsselten Text angehängt wird.

Der bekannteste asymmetrische Verschlüsselungsalgorithmus wurde am MIT von Rivest, Shamir und Adleman entwickelt und deshalb RSA genannt.

Der mathematische Hintergrund (Beispiel einer asymetrischen Verschlüsselung)

Wie, fragt man sich, kann so etwas funktionieren? Wie ist es möglich, daß man ein Schlüsselpaar so benutzen kann, daß eine Chiffretext, der mit dem einen Schlüssel erzeugt wurde, nur mit dem dazu passenden Pendent wieder entschlüsselt werden kann?

Der folgende kleine Ausflug in die Mathematik soll das ein wenig plausibel machen. Auch wenn hier kein vollständiges mathematisches Modell entwickelt wird, hilft es zumindest zu verstehen, worum es bei der asymmetrischen Verschlüsselung geht.

Wir nehmen eine Funktion φ(n) an. Diese Funktion liefert die Anzahl derjenigen natürlichen Zahlen, die zu n teilerfremd und nicht größer als n sind. Ein Beispiel: φ(14) = 6. Warum? 1, 3, 5, 9, 11, 13 sind kleiner als 14 und haben keinen gemeinsamen Teiler mit 14. Vereinfacht kann man sagen: Wenn n eine Primzahl ist, dann ist φ(n) = n - 1. Ist n nun aber keine Primzahl, und ist n nur groß genug, steht man bei der Berechnung von φ(n) mit dem Rechner vor einem annähernd unlösbaren Problem.

Euler bewies im 18. Jahrhundert den Satz:

$$b^{\varphi(n)} \bmod n = 1 \qquad \textbf{(GL 1)}$$

unter der Voraussetzung, daß b und n zwei teilerfremde Zahlen sind. Die Funktion mod (Modulo) ist der Rest einer ganzzahligen Division (z.B. 5 mod 2 = 1). Potenziert man diese Gleichung mit k und multipliziert sie anschließend mit b, dann ergibt sich für b < n:

$$b^{k\varphi(n)+1} \bmod n = b \qquad \textbf{(GL 2)}$$

Wir können dann kφ(n) + 1 in zwei Faktoren zerlegen:

$$k\varphi(n) + 1 = e \times d \qquad \textbf{(GL 3)}$$

Setzt man diese Formel ein, hat man im Grunde eine Funktion, die sich zur Verschlüsselung eignet:

$$b^{ed} \bmod n = (b^e \bmod n)^d \bmod n = b \qquad \textbf{(GL 4)}$$

Sei b nun ein Teil der zu verschlüsselnden Nachricht, e und n gemeinsam der öffentliche Schlüssel und d der geheime Schlüssel. Dann kann

$$c = b^e \bmod n \qquad \textbf{(GL 5)}$$

zum Erzeugen des Chiffretextes genutzt werden und

$$b = c^d \bmod n \qquad \textbf{(GL 6)}$$

zur Entschlüsselung. Die Berechnung eines geeigneten Schlüsselpaares ist nicht trivial. n, e und d müssen so beschaffen sein, daß φ(n) schwer zu berechnen ist. Das ist dann der Fall, wenn n keine Primzahl ist. Ich gehe hier nicht näher auf die Generierung geeigneter Schlüssel ein. Es sei nur soviel gesagt: Für das RSA - Verfahren läßt sich beweisen, daß b und n nicht, wie von Euler gefordert, teilerfremd sein müssen [Heg94].

2.3 Einwegfunktionen

Die Einwegfunktionen spielen zusammen mit öffentlichen Schlüsseln eine große Rolle in der Kryptographie. Die Einwegfunktionen lassen sich leicht berechnen, sie besitzen aber keine Umkehrfunktion.

Zur Verschlüsselung kann eine Einwegfunktion nicht verwendet werden, da dann der Empfänger nicht entschlüsseln kann.

Einweg-Hashfunktionen

Einweg-Hashfunktionen sind Grundlage für viele Protokolle. Je nach Verwendung werden sie sehr unterschiedlich genannt:

Kompressionsfunktion, Kontraktionsfunktion, Message-Digest, Fingerabdruck, kryptographische Prüfsumme, Integritätsprüfung von Nachrichten.

Die Hashfunktion nimmt einen Eingabestrom variabler Länge und wandelt diesen in einen Ausgabestrom fester Länge um.

Die Sicherheit dieser Funktion liegt in der Einweg-Eigenschaft. Die Ausgabe ist nicht nachvollziehbar von der Eingabe abhängig. Durch Änderung eines einzigen Bits in der Eingabe ändert sich im Mittel die Hälfte aller Bits im Hashwert. Mit anderen Worten: Es ist extrem aufwendig, einen Eingabestrom zu erzeugen, der einen bestimmten Ausgabestrom erzeugt.

Es ist vom Berechnungsaufwand her undurchführbar, zu einem bestimmten Hashwert das Original zu finden. In der Praxis werden Hashfunktionen mit Hashwerten der Länge 128 Bit benutzt. Für längerfristige Sicherheit wird ein Hashwert mit 160 Bit benutzt.

Die Hashfunktionen werden auf Grundlage einer Kompressionsfunktion entwickelt. Im folgenden ein paar Beispiele:

MD4 (Message Digest) ist eine Einweg-Hashfunktion. Der Algorithmus erzeugt einen 128 Bit langen Hashwert der Eingabenachricht. Das Ergebnis wird dann für die digitale Signatur verwendet. MD4 eignet sich für schnelle Software-Implementierung, da der Algorithmus nur einfache Bitmanipulationen mit 32-Bit-Operanden benutzt.

MD5 ist eine verbesserte, aber auch komplexere Version von MD4. Auch MD5 produziert einen 128 Bit langen Hashwert der Eingabenachricht [Schn96].

2.3.1 Authentifizierung durch asymmetrische Verschlüsselung

Falls die beiden zusammengehörigen Schlüssel eines asymmetrischen Verfahrens austauschbar sind, d.h. es läßt sich auch eine mit dem geheimen Schlüssel kodierte Nachricht nur mit dem zugehörigen öffentlichen Schlüssel dekodieren, kann der Empfänger den Absender einer Nachricht eindeutig identifizieren.

Dabei wird so vorgegangen: Der Absender der Nachricht berechnet einen Hashwert der zu unterzeichnenden Nachricht. Er erhält z.B. einen 128 Bit langen Wert. Diesen verschlüsselt er mit seinem privaten Schlüssel und hängt ihn an die Nachricht. Der Empfänger kann jetzt zugleich zwei Prüfungen vornehmen. Zum einen: Läßt sich der angehängte Hashwert mit dem öffentlichen Schlüssel des Absenders entschlüsseln, ist sichergestellt, daß der Hashwert vom Absender stammt. Jetzt kann der Empfänger selbst den Hashwert nach dem gleichen Algorithmus wie der Absender berechnen und sein Ergebnis mit dem des Absenders vergleichen. Stimmen beide Werte überein, ist die Nachricht vom Absender mit sehr hoher Wahrscheinlichkeit genau so abgesandt worden. Das heißt, der Text wurde nicht manipuliert und der Absender kann den Inhalt nicht abstreiten. Stimmt der Hashwertvergleich nicht überein, hat ein Angreifer entweder versucht, eine falsche Unterschrift unter ein Dokument zu setzen oder der Text wurde auf dem Weg zum Empfänger verändert.

Dadurch kann mittels der asymmetrischen Verschlüsselung auch die von der Datensicherheit geforderte Authentizität und Integrität übermittelter Nachrichten weitgehend zugesichert werden.

Der RSA - Algorithmus geht davon aus, daß die gesamte Nachricht mit dem privaten Schlüssel des Absenders verschlüsselt wird. Das bringt eine nahezu 100%-ige Sicherheit der Integrität der Nachricht.

Da der Algorithmus jedoch sehr aufwendig ist, wird in der Regel nicht die gesamte Nachricht verschlüsselt, sondern nur, wie beschrieben, der Hashwert über diese berechnet, die dann mit dem geheimen Schlüssel des Absenders verschlüsselt als "Digitale Unterschrift" an die Nachricht angehängt wird. Falls auch die Nachricht an sich verschlüsselt werden muß, wird das anschließend mit Hilfe des öffentlichen Schlüssels des Empfängers getan. Das geschieht wiederum, wie oben beschrieben mit einem hybriden Verfahren.

2.3.2 Schlüsselverwaltung

Es wurde schon angedeutet, daß es bei den verschiedenen Verfahren ein Schlüsselverteilungsproblem gibt. Der öffentliche Schlüssel eines asymmetrischen Verfahrens muß allgemein bekannt gegeben werden. Dabei kann ein "Angreifer" evtl. den Schlüssel durch seinen eigenen ersetzen. Dieser Angriff wird in der Fachliteratur im allgemeinen “Man-in-the-Middle-Attack” genannt. Damit ist gezeigt, daß auch die asymmetrischen Verfahren keine vollständige Lösung des Verteilungsproblems beinhalten. Es gibt sehr unterschiedliche Ansätze, wie man dem Verteilungsproblem begegnen kann.

Vertrauenswürdiger Dritter (engl.: trusted third party)

Wir nehmen eine Person an, deren Schlüssel allgemein bekannt ist. Unterschreibt diese Person den öffentlichen Schlüssel einer anderen Person vor der Verteilung, so kann die Echtheit des neu verteilten Schlüssels von den Empfängern geprüft werden. Eine solche Unterschrift wird Zertifikat genannt. Eine Stelle, die sich vornehmlich mit der Zertifizierung von öffentlichen Schlüsseln beschäftigt, wird Zertifizierungsstelle genannt. Man bemerkt schnell, daß dieses Prinzip mit der Bekanntheit des Schlüssels der Zertifizierungsstelle steht und fällt. Das Problem verlagert sich also auf das Verteilungsproblem zwischen der Zertifizierungsstelle und den Benutzern der Zertifikate. In der Regel wird das Problem durch eine große Verteilung des Urzertifikats gelöst. Diese Urzertifikate werden z.B. mit Programmen gemeinsam auf CDs gepreßt und verteilt, an verschiedenen Stellen im Netz verteilt usw.

Ein digitales Zertifikat ist ein Dokument, welches attestiert, daß ein bestimmter öffentlicher Schlüssel zu einer bestimmten Person oder Organisation gehört. X.509 hat sich mittlerweile als Standard für die Zertifizierung digitaler Zertifikate etabliert.

Geheime Schlüssel dürfen einzig und allein ihrem Besitzer bekannt sein. Da diese jedoch beim RSA-Verfahren sehr lang sind (300 Dezimalstellen), kann er sie wohl kaum auswendig lernen. In der Regel werden diese digital gespeichert (z.B. auf seiner Festplatte oder in Chipkarten). Der gespeicherte Schlüssel wird in der Praxis selbst wieder mit einem Paßwort oder einer PIN gesichert.

Netzwerk des Vertrauens (engl.: web of trust)

Dieses Prinzip, das vor allem von Phillip Zimmermann, dem Entwickler von PGP propagiert wird, beruht auf der Tatsache, daß, wenn nur genügend Personen gegenseitig jeweils die Schlüssel der anderen zertifizieren, die Zertifikate und deren Abhängigkeiten so verworren werden, daß diese zwar noch überprüfbar, jedoch nicht mehr zu fälschen sind. Ist ein öffentlicher Schlüssel einer Person also z.B. von 10 anderen Personen unterschrieben und kennt der Empfänger des Schlüssels nur 5 der Unterzeichner, so muß ein Angreifer den Schlüssel abfangen und auch noch die 5 Zertifikate fälschen. Diese können jedoch wieder von anderen unterschrieben worden sein usw. So entsteht eine endlose Kette von Abhängigkeiten, die sich irgendwann nicht mehr berechnen läßt. Der Vorteil: Man ist nicht von der Richtigkeit eines einzelnen Zertifikats abhängig. Der Nachteil: Die Schlüsselverwaltung und die Kontrolle der Zertifikate liegen zum Teil bei den Endanwendern. Es ist ein gewisses Wissen über

Schlüsselverwaltung und Überprüfung sowie eine Interpretation der Rückmeldungen der Programme nötig. Dieser Aufwand ist für viele Personen zu groß. Auch für PGP Schlüssel gibt es mittlerweile Zertifizierungsstellen.

2.3.3 Die Grenzen der Sicherheit

Wie schon an verschiedenen Stellen angedeutet, haben alle Algorithmen und Verfahren nicht zu unterschätzende Schwachstellen. Bei der Verwendung bestimmter kryptographischer Verfahren in einer Software muß der Entwickler diese Schwachstellen im Auge behalten und mit abschätzen. Dabei ist es von Bedeutung, zumindest die wichtigsten Angriffstaktiken zu kennen, die ein Angreifer nutzen kann, um sich so gut wie möglich gegen diese zu schützen.

Bekanntwerden von Schlüsseln

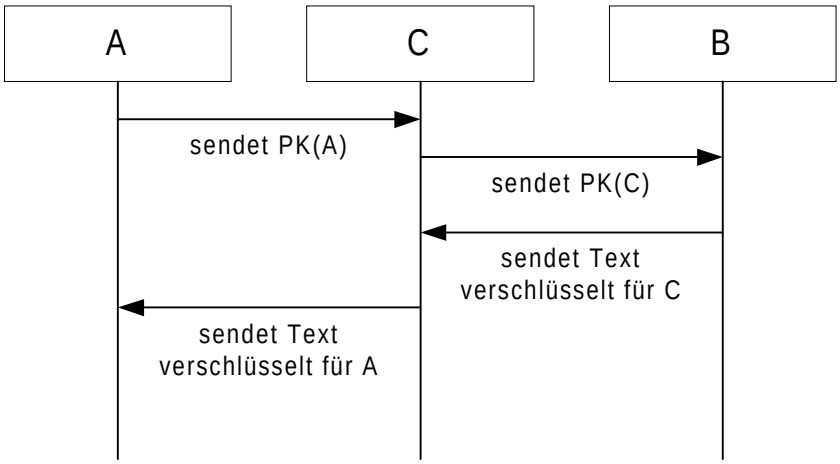
Wenn eine Dechiffrierung eines Textes nur mit Hilfe der entsprechenden privaten Schlüssel möglich ist, liegt die Angriffstechnik nahe, den privaten Schlüssel zu entwenden. Meist ist dieser private Schlüssel deshalb noch durch ein Paßwort oder eine PIN (persönliche Identifikationsnummer) gesichert. Der private Schlüssel wird also mit einem symmetrischen Verfahren verschlüsselt und z.B. auf der Festplatte abgelegt. Da sich ein Mensch jedoch vergleichsweise kurze Paßwörter oder Zahlkombinationen merken kann, ist der Schutz durch diese Mechanismen nur sehr schwach. Es muß also dafür gesorgt werden, daß der private Schlüssel nicht in fremde Hände geraten kann. Einer der besten Möglichkeiten, dies zu gewährleisten, ist die Verwendung von sog. Smartcards (Chipkarten). Diese Smartcards besitzen einen Mikroprozessor, einen eingebauten Verschlüsselungsalgorithmus (ggf. auch Hashfunktionen etc.) und einen kleinen Speicher. Der Speicher ist von außen nur über eine definierte Schnittstelle zugänglich. Die Karte ist in der Lage, einen Schlüssel zu erzeugen, der außerhalb der Karte nicht bekannt werden kann. Will man einen Klartext verschlüsseln, übergibt man der Karte eine PIN und den Text und erhält den Chiffretext von der Karte zurück. Ein Rückschluß auf den Schlüssel ist so kaum möglich. Die Karte ist sowohl in der Lage, ihren öffentlichen Schlüssel mitzuteilen, damit er weitergereicht werden kann als auch eine Anzahl von öffentlichen Schlüsseln und Zertifikaten aufzunehmen. Ferner kann die Karte für eine Sperrung sorgen, wenn öfter als üblicherweise dreimal versucht wurde, mit einer falschen PIN auf die Karte zuzugreifen.

Steht keine Smartcard zur Verfügung, kann der private Schlüssel z.B. auf einer Diskette mitgenommen werden, so daß er einfach physikalisch nicht zugänglich ist. Ein großes Problem stellt auch immer eine Netzwerkverbindung dar, da potentiell immer die Gefahr besteht, daß der Schlüssel auf irgendeinem Weg über das Netzwerk entwendet werden kann. Im Gegensatz zu anderen Diebstählen ist ein solcher Schlüsseldiebstahl kaum festzustellen. Das Opfer weiß also nicht, daß ein Angreifer von diesem Moment an alle Nachrichten entschlüsseln und ggf. auch rechtsverbindliche Unterschriften im Namen des Opfers leisten kann. Das Speichern von privaten Schlüsseln auf UNIX-Systemen ist hier noch schwieriger, da man hier auch noch vom Schutz der Dateien vom Betriebssystem abhängig ist. Werden Laufwerke zusätzlich über das Netzwerk anderen Rechnern zur Verfügung gestellt, ist der Schutz als äußerst schwach anzusehen. Am sichersten ist hier ein nicht von außen zugänglicher Rechner, auf dem die Ver- und Entschlüsselung durchgeführt wird und bei dem über einen Datenträger (nicht über Netzwerkverbindungen) genau die gewünschten Daten nach außen getragen und dann erst in das Netzwerk eingespielt werden. Ein solcher Aufwand ist jedoch selten gerechtfertigt und wird vor allem von Geheimdiensten und militärischen Organisationen eingesetzt, um ihre Schlüssel zu schützen. Im Rahmen dieser Diplomarbeit war ich in der Lage, mich mit Alternativen zu beschäftigen, die ich später noch aufzeigen werde (siehe „Trennung von MH und KH“ auf Seite 34).

Man-in-the-Middle-Attack

Wie bereits erwähnt, ist es möglich, beim Schlüsselaustausch einen Angriff in der Form durchzuführen, daß der Angreifer den zu übertragenden öffentlichen Schlüssel des Absenders abfängt und seinen eigenen Schlüssel an den Empfänger weiterleitet. Von diesem Moment an, ist er in der Lage, alle gesendeten Nachrichten abzufangen, zu entschlüsseln und neu verschlüsselt an den Empfänger zu leiten, ohne daß Sender oder Empfänger das bemerken können, es sei denn, der Angreifer übersieht eine Nachricht, die mit dem Schlüssel des Angreifers verschlüsselt an den Empfänger geht.

ABBILDUNG 5. Timeline Darstellung einer Man-in-the-Middle-Attack



Brute-Force-Attack

Diese Attacke ist ebenso einfach, wie wirksam. Angenommen, es gibt einen Schlüssel bekannter Länge (oder bekannter maximaler Länge), dann ist es möglich, einfach jeden Schlüssel durchzuprobieren und zu testen, ob damit die Nachricht entschlüsselbar ist. Die Einschränkung der bekannten Länge ist nur dadurch gegeben, da man sonst kein Abbruchkriterium hat, bei der der Angriff terminiert werden kann. Mit anderen Worten, man weiß nie, ob man jetzt alle Kombinationen durchprobiert hat oder ob der Schlüssel einfach noch eine Stelle mehr hat, die man durchprobieren muß. Es gibt nur ein wirksames Mittel gegen eine Brute-Force-Attack: Die Anzahl der Möglichkeiten muß so enorm hoch sein, daß ein Durchprobieren in vertretbarer Zeit nicht zu leisten ist. Das ist jedoch extrem schwierig, da die Rechenleistung der neuen Prozessoren so schnell ansteigt, daß man nur schwer mit Schlüssellängen hantieren kann, die auch in vielen Jahren noch schwer zu knacken sind. Mit anderen Worten: Wenn heute ein Schlüssel noch in 1000 Jahren nicht zu knacken ist, ist in einem Jahr ein Rechner vielleicht 1000 mal schneller und schafft eine Brute-Force-Attack innerhalb eines Jahres. Wir müssen also mit Sicherheiten arbeiten, die einige Millionen Jahre nach heutigem Stand der Technik standhalten würden, damit sie realistischerweise 1000 Jahre standhalten. Mit solchen Schlüssellängen zu arbeiten ist aber kaum machbar, da heute Rechner an solchen Verschlüsselungsaufgaben zu lange arbeiten, so daß eine solche Verschlüsselung organisatorisch nicht durchführbar wäre.

2.3.4 Computer-Algorithmen

Es gibt zahlreiche Computer Algorithmen, von denen ich die fünf verbreitesten nachfolgend aufzeigen werde:

DES (Data Encrypton Standard)

DES ist der am häufigsten verwendete Algorithmus zur symmetrischen Verschlüsselung. Er ist sowohl US- als auch internationaler Standard.

Kritiker behaupten, daß dieser Schlüssel zu kurz ist, und unter Umständen mittels sehr hohem Aufwand zu knacken ist. Dennoch ist der Algorithmus wegen seiner Effizienz sehr beliebt.

RSA (Rivest, Shamir und Adelmann)

ist der beliebteste asymmetrische Algorithmus mit öffentlichen und privaten Schlüsseln. Er kann sowohl zur Verschlüsselung als auch zur Signatur verwendet werden.

Er basiert auf der Faktorisierung des Produktes zweier unbekannter, sehr großer Primzahlen, was mathematisch nur sehr aufwendig zu realisieren ist.

DSA (Digital Signatur Algorithm)

Der DSA-Algorithmus wurde vom NIST im Rahmen des DSS veröffentlicht. DSS ist für den elektronischen Unterschriftenstandard in den USA entwickelt worden. Dieser Standard wurde im Mai 1994 herausgegeben.

Die Schlüssellänge kann beliebig sein, wird aber nur zwischen 512 und 1024 Bit eingesetzt. DSA kann laut Spezifikation nur als digitale Unterschrift verwendet werden. Mit DSA-Implementierungen lassen sich aber auch Verschlüsselungen realisieren.

Das Unterschreiben läuft in DSA schneller ab als die Verifikation - hingegen ist es in RSA umgekehrt.

Viele Hard- und Software-Hersteller haben bereits RSA lizenziert, so daß sich DSA wahrscheinlich nicht als Standard für digitale Unterschriften durchsetzen wird.

XOR

ist ein sehr einfacher und auch unsicherer Algorithmus. Der zu verschlüsselnde Text wird mit einem Schlüsselwort durch die XOR Operation chiffriert. Nochmalige Anwendung der Operation mit dem gleichen Schlüssel liefert den Originaltext zurück (symmetrisch).

Trotz seiner Unsicherheit wird dieser Algorithmus in etlichen kommerziellen Softwarepaketen verwendet. Kritiker bezeichnen diesen Algorithmus, wie bereits erwähnt, als "Spielzeug".

IDEA (International Data Encryption Algrorithm)

ist eine Blockchiffrierung und arbeitet mit Klartextblöcken der Länge 64 Bit. Die Schlüssellänge ist 128 Bit. Der gleiche Algorithmus dient sowohl zum Verschlüsseln als auch zum Entschlüsseln. Der Algorithmus baut sich aus der Mischung von Operationen unterschiedlicher algebraischer Gruppen auf. Es werden drei Operationen gemischt, die sich sowohl in Hardware als auch in Software leicht realisieren lassen [Schn96]:

- XOR
- Addition modulo 2^{16}
- Multiplikation modulo $2^{16}+1$

2.4 Protokollierung

Ein Notar bietet u.a. eine Dienstleistung an, die sich darauf beschränkt, den Erhalt von Schriftstücken zu bestätigen und festzuhalten. Dies kann dann in Streitfällen zur Klärung benutzt werden. Der Notar kann bestätigen, daß zu einem bestimmten Zeitpunkt das Schriftstück bereits existiert hat.

In der Informatik werden vor allem Logbücher oder Logdateien (engl.: logfiles, da es sich nicht mehr um Bücher handelt) benutzt, um zu protokollieren, welche Aktivitäten auf dem Rechner zu einer bestimmten Zeit stattgefunden haben. Diese Logdateien können selbstverständlich, wie alle anderen Daten auch, gefälscht oder gelöscht werden. Sichert man Logdateien gegen fremde Zugriffe, können sie im bestimmten Umfang Nachweisfunktionen übernehmen.

Es darf jedoch nicht vergessen werden, daß die meisten Programme Logfiles anlegen, um Fehler in der Konfiguration, den Programmen oder den Protokollen zu finden und nicht, um einen Nachweis zu führen.

Die Vergangenheit hat jedoch gezeigt, daß selbst ungesicherte E-Mails als Beweismittel in manchen Ländern dieser Welt zugelassen werden, wenn entsprechende, unabhängige Logfiles vorliegen, die den Vorgang der E-Mail-Zustellung dokumentieren können.

2.5 Empfangsbestätigung

Eine Nachricht gesichert von Person A zu Person B zu befördern ist die zentrale Aufgabe der Kryptographie. Ein anderes Problem ist die Frage, ob Person B überhaupt eine Nachricht erhalten hat. Diese Frage kann z.B. über eine Empfangsbestätigung geklärt werden. Nach Erhalt der Nachricht von Person B bestätigt diese den Empfang der Nachricht, indem sie eine Nachricht zurücksendet. Bekommt Person A innerhalb eines erwarteten Zeitraums keine Empfangsbestätigung, kann es entweder sein, daß Person B die Nachricht nicht erhalten hat oder die Bestätigung verloren gegangen ist. Voraussetzung ist hier die Kooperationswilligkeit von B. Bestätigt B die Nachricht einfach nicht, hilft nur eine Empfangsbestätigung durch Programme. Aber selbst dann kann B abstreiten, die Nachricht erhalten zu haben. Entschuldigungen wären z.B. “Programmfehler”, “Nachricht war nicht lesbar” usw.

2.6 Verwendete Notation

In dieser Arbeit verwende ich folgende Notation:

Gegeben sei ein Klartext **a**, ein öffentlicher Schlüssel **pk** und ein privater Schlüssel **sk** sowie ein symmetrischer Schlüssel **cs** (für “common secret”). Dann bezeichnet **a.crypt(pk)** den Klartext **a**, der mit dem öffentlichen Schlüssel **pk** verschlüsselt ist. Es wird, wenn nicht anders angegeben davon abstrahiert, daß bei der Implementierung der Text **a** zunächst mit einem symmetrischen Schlüssel kodiert und dieser dann mit dem öffentlichen Schlüssel chiffriert wird. Der Ausdruck **a.sign(sk)** bedeutet, daß der Text **a** mit dem privaten Schlüssel **sk** signiert wurde, also eine Verifizierung mit **pk** möglich ist, sofern **sk** und **pk** zusammengehören. Der Ausdruck **a.crypt(cs)** bedeutet eine symmetrische Verschlüsselung.

2.7 Firewalls

Der alleinige Einsatz von Kryptographie liefert nicht den geforderten Schutz für das zu entwickelnde E-Mail System. Vielmehr müssen wir uns auch um die Sicherheit der Rechner kümmern, auf denen die Daten gespeichert sind und auf denen sie verarbeitet werden. Jeder Rechner, auf den über das Internet zugegriffen werden kann, auf dem also Serverprozesse laufen, die nach außen hin Dienste zur Verfügung stellen, ist potentiell gefährdet.

Umgangsprachlich wird diejenige Person, die sich unberechtigterweise Zugang zu anderen Systemen verschafft, “Hacker” genannt. Ich bevorzuge die Definition nach dem “The on-line hacker Jargon File, version 4.0.0”, bei dem solche Personen “Cracker” genannt werden. Hier die Definition laut Jargon File:

cracker /n./

One who breaks security on a system. Coined ca. 1985 by hackers in defense against journalistic misuse of hacker (q.v., sense 8). An earlier attempt to establish `worm' in this sense around 1981--82 on Usenet was largely a failure.

Use of both these neologisms reflects a strong revulsion against the theft and vandalism perpetrated by cracking rings. While it is expected that any real hacker will have done some playful cracking and knows many of the basic techniques, anyone past larval stage is expected to have outgrown the desire to do so except for immediate, benign, practical reasons (for example, if it's necessary to get around some security in order to get some work done).

Thus, there is far less overlap between hackerdom and crackerdom than the mundane reader misled by sensationalistic journalism might expect. Crackers tend to gather in small, tight-knit, very secretive groups that have little overlap with the huge, open poly-culture this lexicon describes; though crackers often like to describe themselves as hackers, most true hackers consider them a separate and lower form of life.

Ethical considerations aside, hackers figure that anyone who can't imagine a more interesting way to play with their computers than breaking into someone else's has to be pretty losing. Some other reasons crackers are looked down on are discussed in the entries on cracking and phreaking. See also samurai, dark-side hacker, and hacker ethic. For a portrait of the typical teenage cracker, see warez d00dz.

2.7.1 Angriffe gegen Internet - Server

Am häufigsten brachen Cracker in Systeme auf folgenden Wegen ein (diese Aufstellung ist weder vollständig noch nach irgendeinem Kriterium bewertet):

- Ausspionieren von Paßwörtern durch Mitschneiden von ungesicherten Internetverbindungen. Dies betrifft z.B. die Dienste Telnet und Rlogin. Bei diesen Protokollen werden die Benutzer beim Zugang zu den Rechnern in der Regel nach Paßwörtern gefragt, die der Cracker dann über sogenannte Packet-Sniffer (Programme zur Mitprotokollierung von Netzwerkpaketen, eigentlich zum Auffinden von Fehlern aber eben auch für Angriffe geeignet) mitlesen kann. Der Cracker kann sich dann an Stelle dieses Benutzers Zugang zum Rechner verschaffen.
- Fälschen von Absendeadressen / Namen etc. Manche Sicherheitsmechanismen bauen nicht auf die Abfrage von Paßwörtern auf, sondern z.B. auf die Identität eines Rechners, dem vertraut wird. Die Absendeadressen der Rechner können von Crackern gefälscht werden.
- Ausnutzung von Fehlern in bestimmten Programmen. Einige Serverprogramme, die Internetdienste zur Verfügung stellen, müssen ohne Zugriffseinschränkungen auf den Server-Systemen laufen. Beispiel hierfür ist der SMTP-Server. Um Mails lokal den Benutzern zur Verfügung zu stellen, ist

es erforderlich, daß der SMTP-Server als privilegierter Prozeß gestartet wird, der in die Dateien sämtlicher Benutzer zur Auslieferung der E-Mails schreiben kann. Stellt sich nun heraus, daß es bestimmte Fehler in der Software gibt, können diese ggf. dazu benutzt werden, sich Zugang zum System mit privilegiertem Status zu verschaffen. Dies gilt ferner für diverse andere Prozesse auch.

- Trojanische Pferde sind beliebte Mittel, um sich Zugang zu ansonsten sehr sicheren Systemen zu verschaffen. Das Prinzip ist in der Regel das, dem Systemverwalter ein Programm zuzuspielen, das neben seinem Nutzeffekt auch noch eine zweite Funktion hat, nämlich die, eine Hintertür zu öffnen, über die der Cracker in das System gelangen kann.

Dies sind nur einige Beispiele für Möglichkeiten, sich Zugang zu fremden Systemen zu verschaffen. Hinzu kommen im täglichen Interneteinsatz selbstverständlich noch die Gefahren, die durch zerstörerische Maßnahmen von Angreifern ausgehen und nicht den Sinn haben, sich Zugang zu verschaffen, sondern aus welchen Gründen auch immer dafür sorgen, Informationen zu zerstören oder Dienste unbrauchbar zu machen. Man spricht hier dann z.B. von Viren bzw. Denail-of-Service Attacken.

2.7.2 Schutz gegen Angriffe

Die Maßnahmen zum Schutz gegen Angriffe sind ebenso vielseitig, wie die Angriffsmöglichkeiten. Hier findet ein ständiges Wettrüsten zwischen Systembetreibern und Crackern statt. Einige Prinzipien sind jedoch leicht einsichtig und auch relativ einfach umzusetzen.

- Je weniger Dienste ein Rechner zur Verfügung stellt, desto weniger Schwachstellen stehen einem Angreifer zur Verfügung.
- Je kleiner die Anzahl der Personen mit Zugriff zum System ist, desto geringer ist die Wahrscheinlichkeit, daß die Zugangsdaten bekannt werden.
- Unverschlüsselte Zugriffe für Wartungsarbeiten über Internet werden nicht durchgeführt (statt Telnet wird z.B. das neuere und über kryptographische Algorithmen gesicherte SSH benutzt - Vorsicht, es gibt Versionen von SSH, die Fehler beinhalten, die einen Angriff auch hier ermöglichen).
- Es werden nur Programme verwendet, die im Quelltext vorliegen und vom Administrator untersucht werden können, um trojanische Pferde auszuschließen (diese Forderung ist nur sehr schwer umzusetzen). Alternativ muß Software von vertrauenswürdigen Herstellern verwendet werden und eben nur diese.

Ein wirkungsvoller Schutz ist die Verwendung einer Firewall. Die Familie der Firewalls besteht zum einen aus sog. Paket-Filtern und zum anderen aus sog. Application-Gateways.

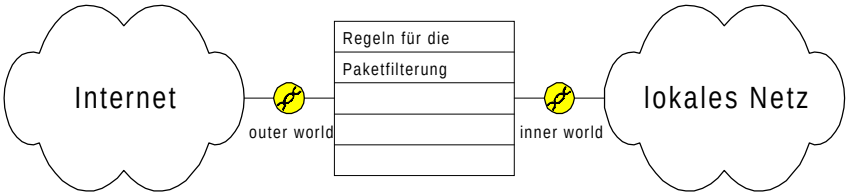
Die Paket-Filter arbeiten auf eine einfache Weise. Im Internet werden Daten in Paketen übermittelt. Jedes Paket besitzt neben diversen anderen Steuerinformationen die IP-Nummer des Absenders, die IP-Nummern des Empfängers und die Portnummer desjenigen Prozesses auf der jeweiligen Maschine, welcher für die Bearbeitung zuständig ist. Mit anderen Worten: Die IP-Nummer identifiziert den Rechner und die Portnummer den Dienst auf diesem Rechner, der genutzt werden soll. Das ist notwendig, weil jeder Rechner verschiedene Dienste zur Verfügung stellen kann und diese eindeutig angesprochen werden können sollen.

Für einige Dienste sind die Portnummern standardisiert. So ist üblicherweise ein SMTP-Server über die Portnummer 25 zu erreichen und ein POP3-Server über die Portnummer 110.

Der Paket-Filter analysiert die Steuerdaten der IP-Pakete und filtert diese nach vorgegebenen Regeln aus. Dazu besitzt ein Paket-Filter in der Regel zwei Netzwerkanschlüsse (z.B. zwei Netzwerkkarten). Ein Anschluß wird als äußerer Anschluß bezeichnet (engl. outer world) und stellt in der Regel die Verbindung zum Internet dar. Der andere Anschluß ist der innere (engl. inner world) und ist die Verbindung zum geschützten Bereich. Um die Sicherheit der Firewall selbst nicht zu gefährden, werden

üblicherweise alle anderen Dienste auf diesem Rechner deaktiviert. Der einzige Zugang findet also über die Konsole statt. Das kann nicht immer so durchgesetzt werden. Oft sind Fernwartungen nötig und nicht immer ist genügend Geld vorhanden, diesen Rechner nur für diese Aufgabe abstellen zu können.

ABBILDUNG 6. Funktionsweise einer Firewall der Klasse Packet-Filter



Konfiguriert werden kann ein Paket-Filter nun wie folgt: Pakete können je nach IP-Nummer und Portnummer am Eingang, am Ausgang und beim Weiterleiten von einem Anschluß zum anderen gefiltert werden. So kann man beispielsweise einstellen, daß kein IP-Paket durch die Firewall geht, das nicht entweder der Portnummer 110 oder 25 zugeordnet ist. Dies läßt sich auch noch von IP-Nummer zu IP-Nummer unterschiedlich konfigurierten.

Application-Gateways verarbeiten nicht nur die Steuerinformationen der IP-Pakete sondern auch deren Inhalt. Dazu ist es erforderlich, das Protokoll der jeweiligen Applikation zu verstehen, um es analysieren zu können. Für jeden Dienst muß also ein eigener Gateway implementiert werden. Application-Gateways sind jedoch sehr viel mächtiger als Paket-Filter. So werde ich in meiner Arbeit zeigen, wie mit einer Art Application-Gateway der E-Mail-Verkehr weiter erlaubt werden und trotzdem jede E-Mail auf Viren geprüft werden kann („Virusschutz“ auf Seite 90).

KAPITEL 3

Bestehende Lösungen

Das folgende Kapitel soll einen kleinen Überblick über bestehende Produkte zum Thema “sichere E-Mail” geben. Dabei möchte ich die Probleme aufzeigen, die die Benutzung der jeweiligen Software mit sich bringen.

3.1 S/MIME basierte Lösungen

Im Rahmen der X.400 und X.500 Spezifikation der ISO wurde auch das Protokoll X.509 festgelegt. Es beschreibt die Verschlüsselung z.B. von E-Mails. Die dort vorgestellten Verfahren können auch auf Internet-E-Mails, die nicht dem X.400 Standard entsprechen, angewendet werden. Der Spezifikation X.509 liegt das Verfahren der "trusted third party", also einer dritten vertrauenswürdigen Person oder Organisation, zugrunde. Eine solche Organisation ist z.B. Verisign. Aber der Betrieb von sicheren Servern und die Verwaltung von Zertifikaten benötigt selbstverständlich hoch qualifiziertes Personal und einen hohen Aufwand an Material. Aus diesem Grund muß ein Schlüssel bei einer solchen Organisation bezahlt werden. Üblicherweise kauft man einen solchen Schlüssel für eine bestimmte Zeit bei einem solchen Anbieter. Dafür wird der öffentliche Schlüssel dort aufgehoben und in die Schlüsselverwaltung eingebracht. Über den Verbleib des privaten Schlüssels hat der Nutzer praktisch keine Kontrolle. Er kann nur hoffen, daß dieser Schlüssel nach der Übertragung vernichtet wird oder sonst keinem zugänglich gemacht wird [X.400][X.500][X.509].

S/MIME ist im RFC 2311 und 2312 spezifiziert und wird heute schon von vielen E-Mail Clients unterstützt. Dazu gehören der Netscape Communicator und Microsoft Outlook, um die am weitesten verbreiteten zu nennen.

Für den Einsatz in der Verwaltung würden folgende Nachteile entstehen: Die Verwaltung wäre auf solche E-Mail Clients beschränkt, die eine S/MIME-Unterstützung anbieten. Es kann nicht mehr von einer freien Wahl der Produkte gesprochen werden. Für jeden Verwaltungsangestellten müßte bei einer Zertifizierungsstelle ein Schlüssel angefertigt werden. Jede personelle Änderung müßte mit einer Umorganisation bei der Zertifizierungsstelle einhergehen. Es gibt keine direkte Unterstützung von Mailinglisten. Ein Schlüssel ist an eine E-Mail Adresse gebunden.

3.2 Pretty Good Privacy (PGP)

Einen ganz anderen Weg geht das Programm PGP (Pretty Good Privacy) von Phillip Zimmermann. Phillip Zimmermann, selbst ein erfahrener Kryptoanalytiker, entwickelte dieses Werkzeug nach dem Konzept des "trust no one". Er geht dabei grundsätzlich davon aus, daß jeder Nutzer seine eigene kleine Datenbank mit den öffentlichen Schlüsseln von potentiellen E-Mail Empfängern hat.

Ein schwieriger Punkt ist, auf eine vertrauenswürdige Art an diese öffentlichen Schlüssel zu gelangen:

PGP nutzt ein Verfahren, das als "web of trust" in der Literatur bekannt ist.

Der beste Weg ist, persönlich vom Empfänger eine Diskette in die Hand zu bekommen, und zu Hause diese Diskette einzulesen. Aber auch ein dritter, schon bekannter Schlüssel kann helfen, an die Daten zu kommen.

Gehen wir davon aus, daß Alice Bob einen Schlüssel zukommen lassen möchte. In diesem Fall reicht es, daß Alice und Bob einen gemeinsamen Bekannten haben, dessen Schlüssel mindestens Bob kennt. Alice übergibt dieser dritten Person nun ihren Schlüssel und läßt diesen unterschreiben. Damit kann Bob indirekt schlußfolgern, daß er den richten Schlüssel erhalten hat.

Steht dies alles nicht zur Verfügung, geht man folgenden Weg: Alice schickt Bob ihren Schlüssel per E-Mail. Danach ruft sie Bob an. Bob läßt den Fingerprint dieses Schlüssels errechnen, Alice tut das gleiche (der Fingerprint ist vergleichbar mit einem Hashwert). Jetzt können sie die Zahlenkolonnen gemeinsam durchgehen und so ziemlich sicher sein, daß es sich hierbei um den korrekten Schlüssel handelt. Der Fingerprint hat hierbei die Eigenschaft, daß es schwer ist, einen zweiten Schlüssel zu erzeugen, der den gleichen Fingerprint hat. Damit wird einer "man-in-the-middle" Attacke vorgebeugt.

PGP geht im Konzept "web of trust" sogar so weit, daß es möglich ist, die Vertrauenswürdigkeit eines Schlüssels direkt anzugeben; d.h. es ist möglich, zu einem Schlüssel anzugeben: "Alle von dieser Person unterschriebenen Schlüssel sind so vertrauenswürdig, als hätte ich sie persönlich bekommen" oder "Alle von dieser Person unterschriebenen Schlüssel müssen noch einmal geprüft werden". Über verschiedene Unterschriften kann dann ein Vertrauenswürdigkeitsgrad berechnet werden [Zim90].

PGP hat die Vorteile, daß es zumindest in der Version 2.x frei zur Verfügung steht. Ein Problem stellt hier nur das US-Gesetz zur Ausfuhr von harter Kryptographie dar. Daher gab es neben der US-Version des Programms lange Zeit noch eine europäische Variante, die weniger sichere Schlüssellängen verwendete. Heute umgeht man dieses Problem, indem man den systemunabhängigen Quellcode in Büchern abdruckt, die nicht den Ausfuhrbestimmungen unterliegen und diese dann außerhalb der USA wieder einscannt bzw. abtippt. Die Fragmente, die nicht zum kryptographischen Algorithmus gehören, können ohnehin ausgeführt werden.

Die 2.x Version von PGP gibt es für viele unterschiedliche Plattformen. Die Ur-Version wurde für das Betriebssystem DOS geschrieben. Von dort aus gab es Portierungen zu nahezu allen UNIX-Derivaten, zum Apple Macintosh und zu zahlreichen anderen Plattformen. Die Benutzungsschnittstelle war jedoch auf nahezu allen Plattformen nicht für die breite Masse konzipiert. PGP 2.x ist lediglich in der Lage, über Kommandozeilen zu operieren und stellt dabei eine simple Schlüsselverwaltung und die Ver-, Entschlüsselung, Signierung und Prüfung von gegebenen Dateien zur Verfügung. Erst Drittanbieter stellten, zumeist als Public Domain[1] (PD) - Programm, eine graphische Schnittstelle oder eine Anbindung an bestehende E-Mail Programme zur Verfügung.

PGP 5.x geht hier einen anderen Weg. Unter Windows 95/NT taucht PGP z.B. in der sog. Traybar (in der Regel am rechten unteren Bildschirmrand) auf und ist so als ständig laufender Prozeß jederzeit verfügbar. Hier ist es möglich, Texte einfach in die Zwischenablage (engl. clipboard) des Betriebssystems zu nehmen (meist durch Anwählen des Textes und Auswahl des Menüpunktes Bearbeiten/

1. Public Domain-Programme sind Programme, die kostenlos verwendet werden können, ohne daß der Benutzer in irgendeiner Form eine Lizenzgebühr zu entrichten hat.

Ausschneiden), sie von dort aus zu verschlüsseln und zu signieren und dann wieder mittels Einfügen in das benutzte Programm zu senden. Aber auch Plugins für Exchange oder Outlook stehen zur Verfügung, um direkt aus den Mailprogrammen eine Verschlüsselung durchzuführen.

Nur gab es gerade um PGP 5.x gewaltige Diskussionen, weil die gegründete Firma nun doch an einigen Stellen gemeinsame Sache mit staatlichen Einrichtungen in den USA zu machen scheint, die den sicheren E-Mail-Verkehr eingeschränkt sehen wollen. Solche Diskussionen laufen unter Bezeichnungen, wie "key recovery". Wer Schlüssel wieder herstellen darf und kann und unter welchen Umständen ist ein viel diskutierter Punkt. Neben den Versionen 2.x und 5.x von PGP gibt es auch eine Variante des Individual Network e.V. (IN), die einige Probleme der PGP 2.x Versionen (die in [Camp98] eingehend beschrieben sind) behebt.

In diesem Zusammenhang ist ferner interessant, daß PGP jetzt standardisiert werden soll. Leider machten die PGP Entwickler jedoch den Fehler, den Standard als RFC einzureichen. Solche RFC haben die grundlegende Eigenschaft, daß sie öffentlich diskutiert werden können. Der IN e.V. hat hier seine eigene Version des RFC zur Diskussion eingebracht. Wohin die Entwicklung von PGP gehen wird, ist ungewiß. Eine eigene Schlüsselverwaltung ist zwar unter verschiedensten Sicherheitsgesichtspunkten die einzige Lösung. Nur möchten die wenigsten E-Mail-Nutzer ihre Schlüsselverwaltung wirklich selbst organisieren und sich umfangreiche Anzahl öffentlicher Schlüssel speichern, abgelaufene oder gestohlene Schlüssel wieder löschen oder als ungültig markieren usw. Aus diesem Problem heraus wurden auch für PGP Zertifizierungsstellen eingerichtet [Camp98].

Technisch gesehen verwendet PGP den IDEA-Algrithmus zur Datenverschlüsselung, RSA mit bis zu 2047 Bit zur Schlüsselverwaltung sowie eine MD5 Einweg-Hashfunktion [Schn96].

Bei der Verwendung von PGP entstehen folgende Probleme: PGP nutzt von sich aus nur in der Version 5.x die Keyserver. In der Version 2.x muß der Benutzer selbst die Schlüssel vom Server laden. Ein Problem stellt ferner die Lizensierung von PGP dar. Es gibt keine sinnvollen Konzepte für Mailinglisten. Private Schlüssel können ausschließlich auf der Festplatte oder Diskette, nicht jedoch z.B. auf Smartcards gehalten werden.

Trotz allem ist zu überlegen, wie PGP auch in dieser Arbeit genutzt werden kann, da es sich hierbei um ein sehr weit verbreitetes Konzept handelt.

3.3 Privacy Enhanced Messages (PEM)

PEM geht einen sehr ähnlichen Weg wie PGP. Auch hier werden Public-Key Verfahren eingesetzt. PEM geht jedoch in einigen Punkten in der Funktionalität weiter als PGP.

PEM wurde geschaffen, um in einer X.500 Umgebung zu arbeiten. Im Gegensatz zu PGP ist PEM als Internet-Standard angenommen worden [Linn93]. Die in PEM benutzten Algorithmen können beliebig ausgetauscht werden. Dafür ist ein Standard vorgesehen, der eine Protokollwahl zuläßt. Zur Zeit arbeitet PEM mit DES im CBC-Modus zur Verschlüsselung von Nachrichten. Zur Authentifizierung wird MD2 oder MD5 benutzt. Zur symmetrischen Schlüsselverwaltung kann entweder DES oder Triple-DES verwendet werden. Zur Public-Key Zertifizierung werden RSA (Schlüssellänge bis 1024 Bit) und X.509 für die Zertifikat-Infrastruktur verwendet.

In neueren Versionen von PEM ist auch die Verwendung von Smartcards vorgesehen. Dieses Vorgehen kann den Sicherheitsstand beträchtlich erhöhen. Mittlerweile kommt auch Secude[1] mit einer bunten Windows 95/NT Oberfläche oder mit entsprechendem Design auf verschiedenen Plattformen. Neben dem Secude-Programm steht auch eine C-API zur Verfügung. Secude kann also in eigene Sicherheitsmechanismen eingebaut werden.

PEM kommt aus meiner Sicht einer Problemlösung am nächsten. Ferner ergibt sich der glückliche Zufall, daß die GMD Darmstadt ein Projektpartner beim E2S Projekt war und damit der TU eine kostenlose Secude-Lizenz zur Verfügung steht (zumindest im Rahmen des Projektes, was auf diese Arbeit zutrifft). Probleme entstehen jedoch dadurch, daß die Identifizierer der Schlüssel im X.509 Format angegeben sind (z.B. C=de, O=TU-Berlin, CN=Thomas.Hildmann) und auch hier kein brauchbares Konzept für Mailinglisten zur Verfügung steht.

3.4 Virtual Private Networks

Es kommt vor, daß in einem Netz Produkte eingesetzt werden, die selbst keine eigene Verschlüsselung und / oder Authentifizierung beherrschen, jedoch schützenswerte Daten übertragen. In diesem Fall muß man die Absicherung auf tieferen Schichten der Übertragung übernehmen. An dieser Stelle greifen sog. Virtual Private Networks (VPN). Sie können z.B. eingesetzt werden, wenn ein WWW-Server örtlich von einer Datenbank getrennt ist, von der er über Internet Daten bezieht oder zwei verteilte Datenbanken über Internet Daten austauschen müssen. VPNs sind hier in der Lage, die Datenintegrität und die Vertraulichkeit zu gewährleisten. Das Prinzip ist denkbar einfach: Zwei Knoten bauen über Internet eine sichere Verbindung auf, nachdem sie sich gegenseitig authentifiziert haben. Über den abgesicherten Kanal wird erneut eine IP-Verbindung etabliert, die über andere Adressen geroutet wird (hierzu steht zum Beispiel das Netz 10.0.0.0 zur Verfügung). Alle über dieses neue Netz gerouteten IP-Pakete können als sicher angenommen werden. Es gibt diverse Produkte zum Thema VPN. Eines der bekanntesten Produkte ist PPTP [Kos98].

Ein ganz einfaches VPN läßt sich mit PD-Software aufbauen. Man etabliert eine SSH-Verbindung zwischen zwei Rechnern. Nun startet man auf beiden Seiten ein PPP oder SLIP, über das man wiederum IP-Pakete übertragen kann. Diese Lösung ist nicht besonders perfomant oder elegant, aber sie funktioniert [Holz98].

Nun wäre es denkbar, das gesamte TU-Verwaltungsnetz mit einem VPN auf Basis von SSH zu verbinden. Teile könnten über das Internet gehen. E-Mails wären dann durch das VPN geschützt. Nur ist dieser Schutz leider nur auf Angriffe von außen bezogen. Ist im Verwaltungsnetz ein Angreifer, hat dieser freie Hand. Die Benutzer des E-Mailsystems hätten in diesem Fall jedoch nichts mit der Handhabung von Schlüsseln oder sonstigen Einschränkungen zu tun. Ein anderes Problem wäre, daß jeder auf der Dienstreise befindliche Professor, der auch am sicheren E-Mail-System teilnehmen will (z.B. über seinen Laptop), mindestens noch einen Rechner mitnehmen müßte, der das VPN etablieren kann. Denn das SSH-VPN läßt sich mangels passender Software unter Windows und anderen Systemen zur Zeit nur über UNIX-Derivate aufbauen. Alternativ könnte ein kostenpflichtiges VPN-Produkt erworben werden, das für alle Plattformen erhältlich ist.

Neben dem sehr geringen Schutz, den das VPN in diesem Fall bieten würde, - es bietet nur Schutz gegen Angriffe von außen, nicht aber von innen - würde dem Vorhaben auch ein hoher Preis für die VPN-Software und diverse Firewalls entgegenstehen.

1. Secude ist ein von der GMD entwickeltes Sicherheitsprodukt, das u.a. auch PEM beherrscht. Es enthält ferner eine Programmbibliothek, die zur Erstellung eigener Programme mit kryptographischen Funktionen dient.

KAPITEL 4

Automatisierung von sicherem elektronischen Schriftverkehr

4.1 Grundprinzipien

Wie wir festgestellt haben, gibt es verschiedene Produkte auf dem Markt, die einen sicheren E-Mail Verkehr ermöglichen. Das Wort “sicher” steht dabei immer für: E-Mail Verkehr mit kryptographischer Absicherung und bietet eine gewisse Sicherheit, wie bereits diskutiert. Alle diese Produkte beschränken sich jedoch auf den Austausch von E-Mails zwischen zwei Partnern. Sollen Mails an eine Gruppe von Personen geschickt werden, geht dies nur über Angabe jedes Individuums und nur dann, wenn jeder Schlüssel bekannt ist oder über einen Keyserver bezogen werden kann. Alternativ gäbe es die Möglichkeit, einen Schlüssel für eine ganze Gruppe zu generieren und den privaten Schlüssel-Teil an alle Mitglieder dieser Gruppe zu verteilen. Probleme gibt es dann selbstverständlich, wenn eine Person die Gruppe verläßt. Dann müßte ein neuer Schlüssel generiert werden und der alte würde seine Gültigkeit verlieren. Bei nur ausreichend komplexen Gruppenzusammensetzungen kann das einen erheblichen Verwaltungsaufwand bedeuten. Ferner ist ein solches Vorgehen für die Sender von Nachrichten wenig transparent und sehr fehleranfällig.

Unser Projektansatz war nun, nicht nur den sicheren E-Mail Austausch zuzulassen, sondern auch die Benutzer völlig von der Schlüsselverwaltung zu befreien und dabei den Verwaltungsaufwand möglichst gering zu halten.

Wir nennen das erarbeitete Konzept: Sichere Mail Umgebung (engl.: Secure Mail Environment, Abk.: SME). Dieses SME beinhaltet einige Komponenten, die aus verschiedenen Überlegungen entstanden sind.

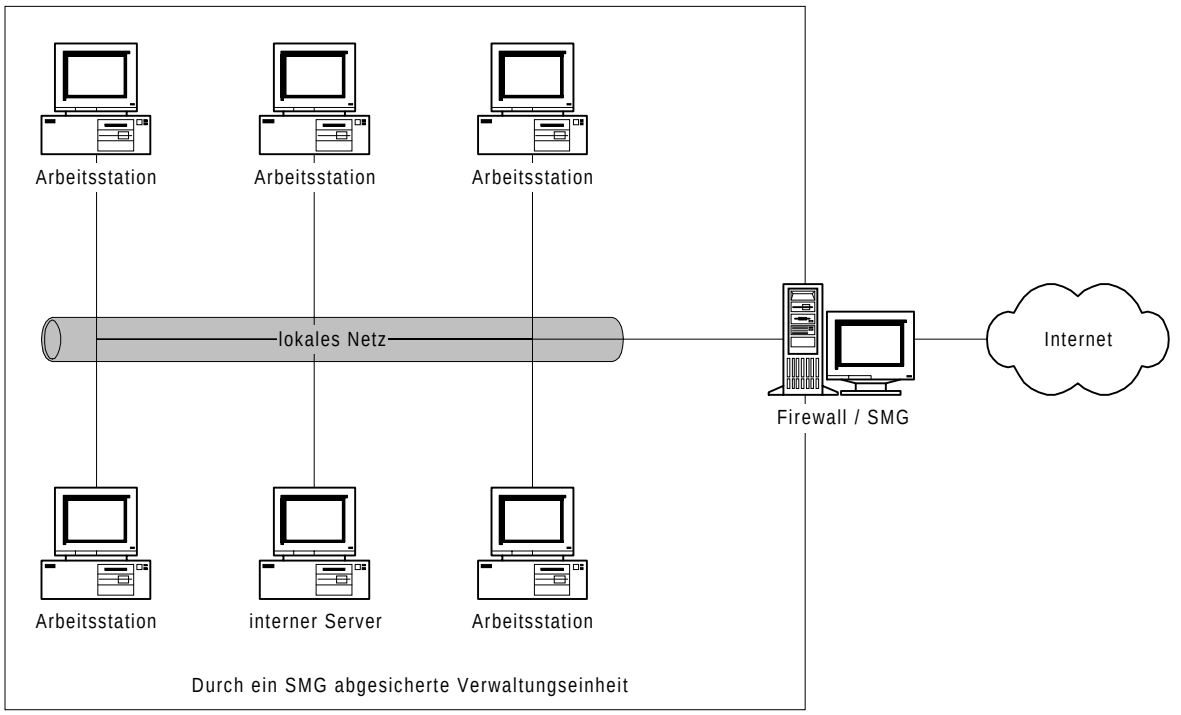
4.1.1 Das Secure Mail Gateway (SMG)

Die erste Idee im E2S-Projekt war die Einführung eines sog. Secure Mail Gateways[1] (SMG). Die Überlegung war: Jedes Institut, jede Verwaltungseinheit ist an irgendeiner Stelle über einen Router mit dem Internet verbunden. Der Router unterliegt an dieser Stelle der Aufsicht der TU. Aus bereits diskutierten Gründen müssen die Router ohnehin mindestens zu Paket-Filter Firewalls ausgebaut werden. Wenn wir nun ein Application-Gateway bauen, das jede eingehende E-Mail entschlüsselt und jede aus-

1. Die Begriffsgebung war nicht immer eindeutig. Vor allem am Anfang des Projektes kamen diverse Bezeichnungen durcheinander, weil von unterschiedlichen Entwicklungen und Ideen die Rede war. Ich benutze hier nur die letzten und mittlerweile eindeutigen Namen der einzelnen Komponenten. In älteren E2S-Dokumenten findet man aber Bezeichnungen, wie “Mailexploder” (MEX) o.ä.

gehende E-Mail verschlüsselt, dann bekommt der Benutzer hinter dem SMG nichts von der sicheren Mailübertragung mit. Die Schlüssel könnten an einer zentralen Stelle verwaltet werden und müßten jeweils nur auf den SMG-Rechnern aufgefrischt werden. Ferner wurde eingeführt, daß das Voranstellen eines Unterstrichs ("_") vor die E-Mail Adresse dafür sorgt, daß die E-Mail unverschlüsselt die Einheit verläßt. Dies bietet den Benutzern die Möglichkeit, selbstständig zu entscheiden, welche E-Mails verschlüsselt und welche unverschlüsselt übertragen werden sollen. Nur so kann (leider ungeschützt) z.B. mit anderen Universitäten etc. kommuniziert werden. Innerhalb der Einheit, kann die E-Mail unverschlüsselt übertragen werden, da ohnehin jede Mitarbeiterin und jeder Mitarbeiter in diesem Teilnetz die gleichen Aufgaben zu bewältigen hat und somit berechtigt ist, auf die Daten zuzugreifen. Wir sprechen hier von sog. sicheren Inseln. Auf dem SMG können dann auch zusätzliche E-Mail Adressen eingerichtet werden, die das Adressieren von Stellenzeichen zuläßt. So könnte ich meinem Sachbearbeiter in der Personalstelle eine E-Mail schreiben, ohne mich persönlich an ein Individuum zu wenden (dieser könnte ja gerade im Urlaub sein oder evtl. krank oder versetzt) [Nag96].

ABBILDUNG 7. Darstellung einer "sicheren Insel" mit einem Secure Mail Gateway (SMG)



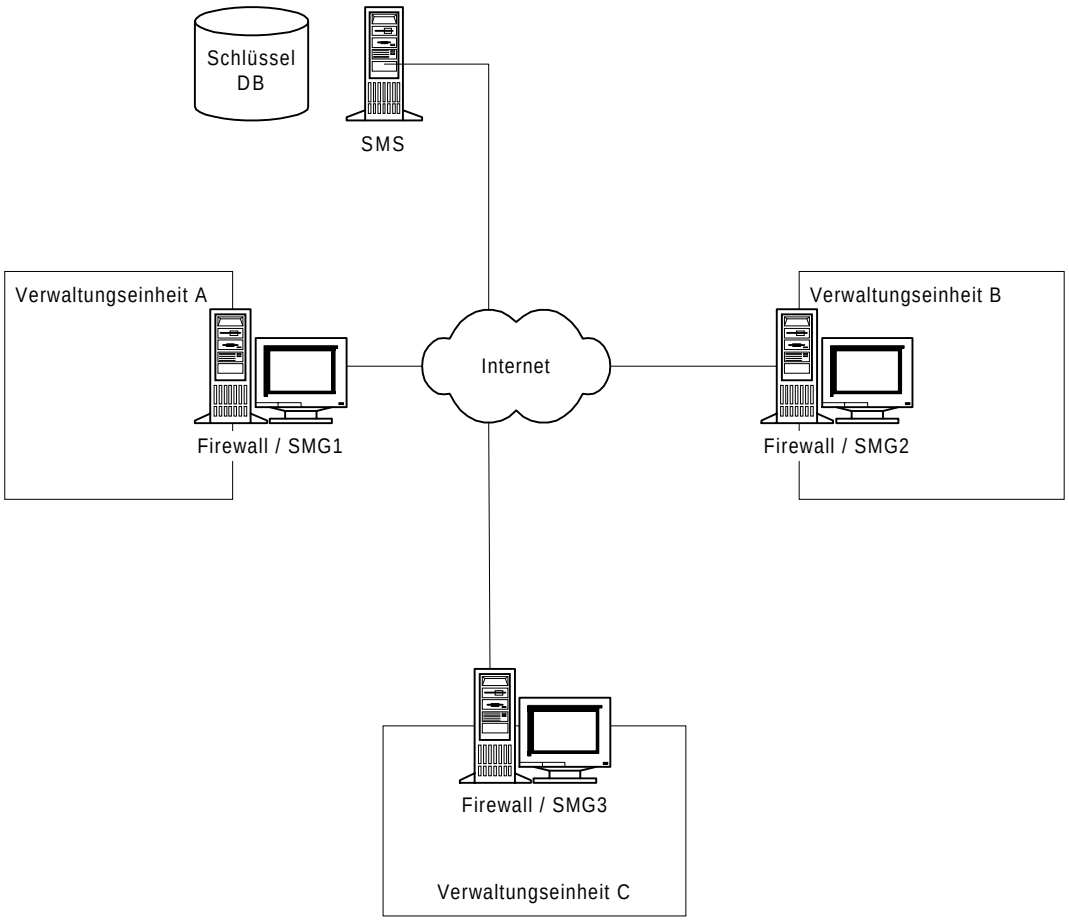
Schwachstelle bei diesem Verfahren ist zum einen das Auffrischen aller Daten sämtlicher SMGs, was bei steigender Zahl von SMGs sehr aufwendig werden könnte und zum anderen die Tatsache, daß innerhalb des SME immer noch Angriffe möglich sind. So muß ein Angreifer aus dem SME nur die Empfängeradresse einer E-Mail fälschen und seine eigene einsetzen. Sofort wird ihm diese E-Mail persönlich zugestellt. Die Entschlüsselung übernimmt sein SMG. Der Verwaltungsaufwand wird noch höher, wenn für Rundschreiben auch Gruppenschlüssel verteilt und gewartet werden müßten (z.B. ein Schlüssel für alle wissenschaftlichen MitarbeiterInnen, der bei jeder personellen Veränderung neu generiert werden müßte).

Um das Problem der Verteilung zu minimieren und eine zentralere Verwaltung zu organisieren und auch um die Hardwareanforderungen an die SMGs weiter herunterzusetzen, brachten wir ferner die Idee eines zentralen Mailservers auf.

4.1.2 Secure Mail Server (SMS)

Der SMS übernimmt einen Teil der Aufgaben der alten SMGs. Die SMGs sollen nur noch den Schlüssel des SMS kennen und ihren eigenen. Der SMS wiederum besitzt die einzig komplette Datenbank der Schlüssel der gesamten Organisation. Ferner verwaltet der SMS auch alle Mailinglisten und zusätzliche Mailadressen der einzelnen Personen sowie die Abbildung der E-Mail Adressen auf die Identifikatoren der Schlüssel. Wir erinnern uns: PEM lag in vielen Belangen sehr dicht an unseren Vorstellungen einer Verschlüsselungssoftware, nur waren die Identifikatoren leider auf X.509 abgestimmt und nicht auf Internet Mailadressen. Eine Abbildung von Internet Mailadressen auf X.509 Adressen kann in einer Datenbank auf dem SMS erfolgen.

ABBILDUNG 8. Mögliches Netzwerk, bestehend aus einem SMS und 3 Verwaltungseinheiten mit SMG



Der Weg einer E-Mail würde also so aussehen:

Person A schreibt einen Text T und schickt diesen an Person B. Person A sitzt hinter dem SMG1 und Person B hinter dem SMG2.

SMG1 empfängt die Mail zur Weiterleitung an den SMS. SMG1 wird also den Text T mit dem öffentlichen Schlüssel des SMS verschlüsseln und ihn signieren.

$$T.crypt(SMS).sign(SMG1) \quad \textbf{(GL 7)}$$

Der SMS kann nun die Signatur prüfen und den Text T wieder herstellen. Danach kann der SMS feststellen, daß Person B hinter SMG2 erreichbar ist und verschlüsselt und signiert selbst den Text.

T.crypt(SMG2).sign(SMS) **(GL 8)**

Geht die E-Mail nicht nur an Person B sondern an beliebig viele anderen Personen, dann kann der SMS die Mail entsprechend oft dublizieren und den Vorgang des Verschlüsselns und Signierens für jeden Empfänger erneut durchführen.

Optimieren ließe sich dieses Verfahren beispielsweise noch dadurch, daß vorher überprüft werden könnte, welche Adressen alle den gleichen Schlüssel haben und an diese nur eine verschlüsselte und signierte E-Mail zu schicken. Für unseren Prototypen reicht die einfache Variante aber zunächst aus.

4.1.3 Header-Signaturen

Es bleibt jedoch das Risiko, daß ein SME-Benutzer die Empfängeradresse fälscht. Ebenso könnte es störend sein, daß die Absenderadresse gefälscht werden kann. So brachte ich die Idee der Header-Signaturen ein.

Eine E-Mail besteht im Internet aus einem Mail-Header nach RFC 822 und einem Body. Header und Body sind durch eine Leerzeile voneinander getrennt. Der Header darf keine Leerzeilen enthalten. Was im Header stehen darf, ist genau definiert. Header-Zeilen haben das Aussehen:

```
<Bezeichner>: <Wert>
```

z.B.

```
To: Thomas Hildmann <hildmann@prz.tu-berlin.de>
```

Es besteht jedoch die definierte Möglichkeit, weitere Zeilen hinzuzufügen. Diese Zeilen tragen nach Definition den Präfix “X-” und haben sonst die gleiche Form, wie die anderen Zeilen. Ferner besteht die Möglichkeit, eine Header-Zeile auch über mehrere tatsächliche Zeilen zu erweitern, sofern die nächste Zeile stets mit einem Leerfeld oder einem Tabulator-Zeichen beginnt.

Diese Tatsache mache ich mir zu Nutze. Ich definiere eine Zeile “X-Signature”. Jetzt trage ich den Absender (“From:”), den Empfänger (“To:”), die Betreffzeile (“Subject:”) und das Datum (“Date:”) zusammen und unterschreibe diese Zeilen z.B. mittels PEM. Das Ergebnis könnte z.B. so aussehen:

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,MIC-CLEAR
Content-Domain: RFC822
Originator-ID-Asymmetric:
 MD8xDjAMBgNVBAUTBVNZUzAwMQwwCgYDVQQDEwNTTVMxEjAQBgNVBAoTCVRVLUJl
 cmxpbjELMAkGA1UEBhMCREU=,00
MIC-Info: RSA-MD5,RSA,
 RRQ0gsF4rmYwB9mkfgUoRM56WVLHHhNd/LJoXm5tk7F2AwNAVcYHMUFGXrSNTWU5
 9aW+WjNMHDtFxyqsoAKEtQ==

Date: Mon, 4 May 1998 22:09:33 +0200
```

```
From: "Klaus Nagel" <99999@hoogla.prz.tu-berlin.de>
To: "Thomas Hildmann" <hildi@sms.tu-berlin.de>
Subject: Re: Mein Test von den SPT

-----END PRIVACY-ENHANCED MESSAGE-----
```

Einige Informationen können hier weggefiltert werden, da diese sich nicht ändern bzw. beim Empfänger zum Verifizieren leicht rekonstruiert werden können. Die verbliebenen Zeilen werden dem RFC 822 entsprechend als “X-Signature:” zusammengesetzt, so daß wir folgendes Ergebnis erhalten:

```
Date: Mon, 4 May 1998 22:09:33 +0200
From: "Klaus Nagel" <99999@hoogla.prz.tu-berlin.de>
To: "Thomas Hildmann" <hildi@sms.tu-berlin.de>
Subject: Re: Mein Test von den SPT
X-Signature: Proc-Type: 4,MIC-CLEAR
   Content-Domain: RFC822
   Originator-ID-Asymmetric:
     MD8xDjAMBgNVBAUTBVNZUzAwMQwwCgYDVQQDEw
     NTTVMxEjAQBgNVBAoTCVRVLUJl
     cmxpbjELMAkGA1UEBhMCREU=,00
   MIC-Info: RSA-MD5,RSA,
     Q1Ym0E0jHd5H5wPUD3NA4g7qiaO33xpwkivcgu3b4yr
     m8LDumVG9HuNpUiihZT8M
     Ze8C5R2g7qqPBLR2F3MTUw==
```

Wird die E-Mail nicht an ein dem SME angehöriges System geleitet, würde die Header-Signatur ignoriert werden. Ansonsten entspricht die Mail jedoch dem PEM Standard. Solange sich die E-Mail jedoch innerhalb des SME befindet, kann die Signatur des Headers benutzt werden.

4.1.4 Normalisierung des Mailheaders

Das RFC 822 definiert zwar das Aussehen des Headers, läßt jedoch innerhalb des Standards diverse Notationen zu. So werden folgende E-Mail Adressen gleich behandelt:

```
<hildmann@prz.tu-berlin.de>
hildmann@prz.tu-berlin.de (Thomas Hildmann)
Thomas <hildmann@prz.tu-berlin.de> Hildmann
```

Ferner wird nicht sichergestellt, daß beim Empfänger die gleiche Form vorliegen muß wie beim Absender. Nur ein Beispiel hierzu:

Wird auf einem UNIX-System beispielsweise mit Hilfe des elm(1) Programms eine E-Mail an einen Benutzer geschickt, der ebenfalls einen Arbeitsbereich auf diesem System hat, so ersetzt elm(1) selbst

die Benutzerkennung durch Benutzerkennung und vollständigen Namen (“hildmann” wird zu “hildmann (Thomas Hildmann)”).

Hinzu kommt die Tatsache, daß selbst die Bezeichner der Header-Zeilen beliebig groß oder klein geschrieben werden können.

Wir führen also eine normalisierte Form des zu überprüfenden Headers ein, die nur die für uns relevanten Daten in einer definierten Form zur Verfügung stellt. Wird die Normalisierung bei Sender und Empfänger auf die gleiche Weise durchgeführt, können die Ergebnisse auch von den kryptographischen Bibliotheken verglichen werden, ohne daß ein anderer Hashwert entsteht.

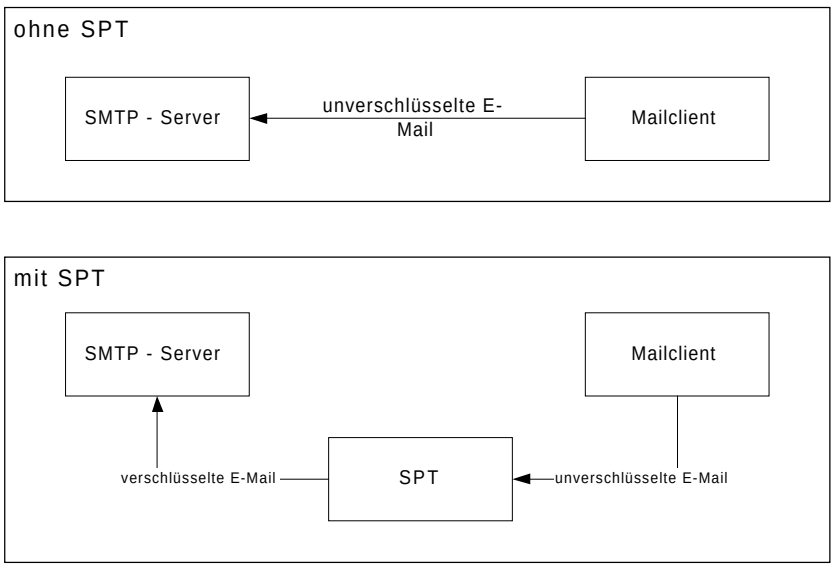
4.1.5 Secure Proxy Tools (SPT)

Bei unserer jetzigen Architektur fehlt noch immer die Möglichkeit, Rechner, die nicht hinter einem SMG stehen einzubinden und noch immer ist es möglich, innerhalb einer Gemeinschaft hinter einem SMG Angriffe gegeneinander durchzuführen. Innerhalb der TU sieht das Modell vor, SMGs genau dort zuzulassen, wo im LAN nur Sachbearbeiterinnen und Sachbearbeiter mit den gleichen Aufgaben sitzen, eine gegenseitige Vertraulichkeit also auch rein rechtlich nicht gegeben sein muß. Wir brauchen also ein Werkzeug, mit dem ein einzelner PC abgesichert werden kann. Eine solche Möglichkeit zu schaffen, lag wiederum in meinem Aufgabenbereich im E2S-Projekt. Geht es darum, eine einzige UNIX-Workstation abzusichern, kann diese so konfiguriert werden, als wäre sie ein SMG. Das wird klar, wenn ich etwas später auf die Implementierung der SMG eingehe.

Das Prinzip der SPT ist denkbar einfach: Mailclients auf PCs arbeiten praktisch alle auf SMTP- und POP3-Basis. Das heißt, daß sie im Falle eines Sendevorgangs eine Internet-Socket-Verbindung zum SMTP-Server aufbauen, mittels SMTP die E-Mail übertragen und die Verbindung wieder abbauen. Analog wird der Empfang realisiert.

Meine Idee war nun, ein Programm zwischen Mailclient und SMTP-Server zu schalten, das grundsätzlich alle Protokolldaten transparent weiterleitet, jedoch bei der eigentlichen Übertragung der E-Mail die Daten vorher verschlüsselt und signiert. Wie bei Proxy-Programmen üblich, sieht das SPT aus Sicht der SMTP-Servers so aus, wie ein normaler Mailclient und aus Sicht des Mailclients sieht der SPT aus, wie ein SMTP-Server. Beide Teile ändern ihr Verhalten also nicht (Prinzip der Proxy-Architektur [Busch98] und [Gam96]).

ABBILDUNG 9. Weg der E-Mail von Mailcient zum SMTP-Server mit und ohne SPT



Leider entstanden aus dieser Idee nun weitere Probleme:

1. Viele der Funktionen mußten jetzt nicht nur auf UNIX-Systemen lauffähig sein, sondern auch auf Windows-Rechnern.
2. Windows PCs sind potentiell gefährdet, von Crackern angegriffen zu werden.
3. Es mußten Möglichkeiten entwickelt werden, sich zwischen Mailclient und SMTP bzw. POP3-Server zu "hängen" und im richtigen Moment die kryptographischen Funktionen zu benutzen.
4. Windows-Benutzer fordern eine graphische Benutzungsoberfläche für ihre Programme. Neben der Entwicklung von Netzwerkkomponenten kam jetzt also auch die Entwicklung einer einfachen Benutzungsoberfläche (engl. graphical user interface, Abk.: GUI) hinzu.

4.2 Einbindung von rollenbasierter Zugriffskontrolle

Im Rahmen seiner Diplomarbeit [Bart98] entwickelte Jörg Bartholdt das Access Control Model. Die im Rahmen dieser Arbeit entworfene Modellierungssprache gestattet es, Firmen- und Organisationsstrukturen und darin enthaltene Rollen abzubilden. Dabei können sowohl Hierarchien, als auch Arbeitsgruppen, Abteilungen usw. abgebildet werden. Rechte, wie z.B. Zugriffsrechte auf Dateien oder ganze Datenbanken können dann auf Rollen anstelle von Benutzern verteilt werden. Bei strukturellen oder personellen Änderungen in der Organisation kann der administrative Aufwand und die Fehleranfälligkeit stark gesenkt werden. Wird zum Beispiel eine Person befördert, so wird ihr eine neue Rolle zugewiesen, über die sie automatisch geänderte Rechte auf einer Vielzahl von Diensten haben kann.

Das Modell arbeitet objektorientiert und kennt neben Objekten auch Benutzer als Spezialfall von Objekten. An Objekte sind einige Standard-Attribute gebunden und einige optionale Attribute. Zwei Attribute spielen im Zusammenspiel der sicheren E-Mail Umgebung mit dem Access Control Model eine wichtige Rolle.

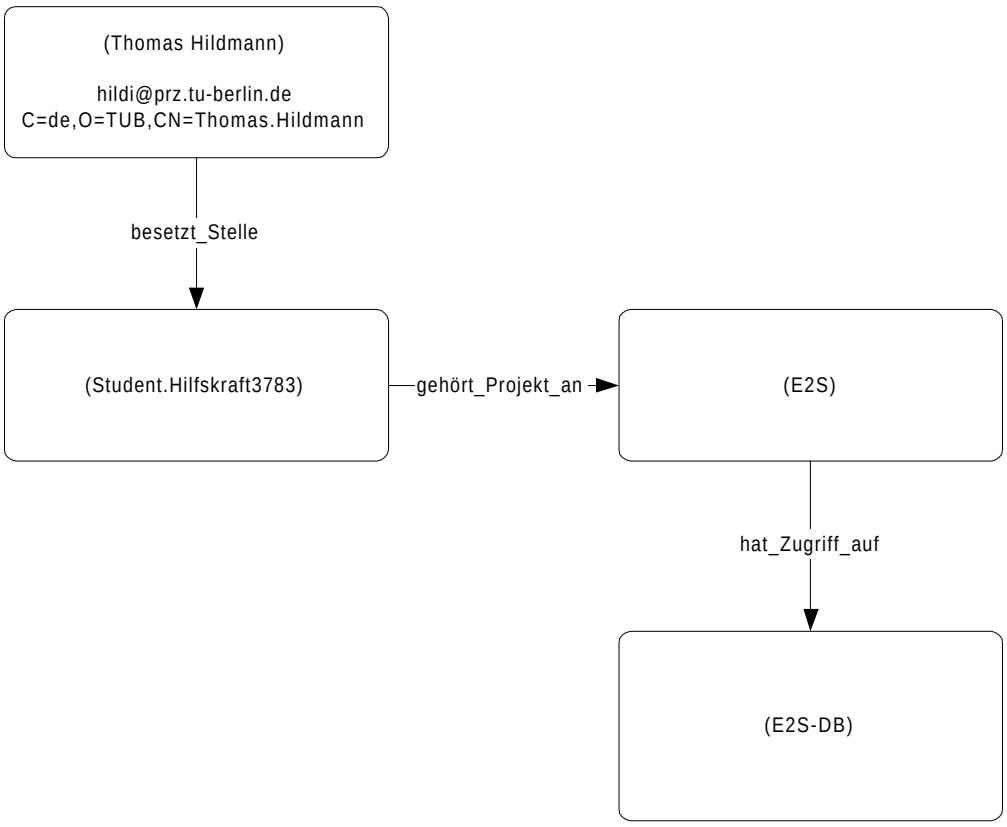
1. Eine E-Mail Adresse, die einem Objekt, in der Regel einem Benutzer zugeordnet werden kann und
2. ein eindeutiger Identifizierer des öffentlichen Schlüssels einer Person oder Gruppe.

Aus dieser Kombination ergeben sich viele Möglichkeiten. Das System kann Personen oder Gruppen unabhängig Adressen und Schlüssel zuordnen. Somit kann für eine Gruppe von Personen der gleiche Schlüssel verwendet werden oder auch eine E-Mail Adresse mittels verschiedener logischer Namen mit verschiedenen Schlüsseln angesprochen werden.

Die grundlegende Idee hinter der Arbeit von Jörg Bartholdt war die, daß für den Zugang zu bestimmten Internetdiensten, wie z.B. Datenbanken mit WWW-Schnittstelle nicht jede Mitarbeiterin und jeder Mitarbeiter einzeln eine Zugriffsberechtigung bekommen muß. Die Probleme liegen auf der Hand. Ändert sich der Status einer Mitarbeiterin / eines Mitarbeiters, so muß diese Statusänderung in allen Datenbanken und sonstigen über z.B. Paßwort geschützten Diensten geändert werden. Stattdessen soll mittels des prototypisch implementierten ACM jede Person ihren Aufgabenbereichen zugeordnet werden. Die Aufgabenbereiche bzw. Stellen werden dann wieder den Diensten und den jeweiligen Zugangsberechtigungen zugeordnet.

Jede Person bekleidet also eine Stelle. Die Stelle ist bestimmten Arbeitsgruppen zugeordnet oder bekleidet z.B. verschiedene Ämter. Die Arbeitsgruppen-Mitglieder oder Ämter haben dann bestimmte Rechte und Pflichten, die mit Hilfe des ACM abgebildet werden können.

ABBILDUNG 10. Beispiel eines Zugriffs auf einen Dienst über Rollen und Mitgliedschaften



Ein solches Modell stellt eine Art Datenbank dar. Wie bereits beschrieben, benötigt unser SMS eine Datenbank, die mindestens eine Verwaltung der Schlüssel ermöglicht. Der Access Control Model Interpreter (ACMI) ist in der Lage, Anfragen bezüglich Objekten innerhalb des ACM zu beantworten.

Das ACMI stellt ein Protokoll zur Abfrage über Netzwerk zur Verfügung. Dies ermöglicht eine Schnittstelle zwischen ACMI und SMS. Durch die Zusammenarbeit der Komponenten ergeben sich folgende Möglichkeiten:

- Zum einen ist das Problem der Schlüsselverwaltung insofern gelöst, als daß nun Schlüssel-Indentifizierer unabhängig von E-Mail-Adressen sind. Ein ACM-Objekt kann einfach einen Identifizierer für einen Schlüssel enthalten.
- Zum anderen ist es möglich, wie gefordert, an Stellen oder Ämter von Personen zu schreiben. Nur ist es nicht mehr nötig, die sog. Alias-Listen, die einer Person verschiedene E-Mail-Adressen zuordnet, zu verwalten. Vielmehr kann diese Verwaltung über das ACM gemacht werden. Benutzt eine Organisation sowohl das ACM als auch das SME, müssen sie ihre Strukturen nur einmal modellieren. Benutzen sie nur das eine oder das andere, stehen ihnen die Modellierungsmöglichkeiten des ACM zur Verfügung.
- Sobald eine Arbeitsgruppe über das ACM modelliert ist, kann diese durch ihren Objektnamen angeschrieben werden.
- Stellvertreterschaften können über das ACM modelliert werden.
- Sind Stellen mit Arbeitsgruppen verbunden und ändert sich die Person, die diese Stelle bekleidet, muß nur der Name der Person im ACM ausgetauscht werden, alle anderen Abhängigkeiten ändern sich implizit automatisch mit.
- Die entstehenden Möglichkeiten gehen weit über das hinaus, was heute unter Mailinglisten verstanden wird!

Es ist möglich, mehrere SMEs miteinander zu vernetzen. Angenommen, es existieren zwei SME mit jeweils einem SMS (SMS A und SMS B), dann können alle Personen von SMS B im ACM von SMS A so eingetragen werden, daß sie an SMS B gehen und für diesen verschlüsselt sind. SMS B verteilt die so empfangenden E-Mails wie gewohnt. Dabei spielt es keine Rolle, ob sie von SMS A kamen oder aus dem lokalen Netz. Das ermöglicht ein Netz von SME-Netzen (siehe hierzu auch „Hierarchisches Domänen-Konzept“ auf Seite 137).

4.3 Umverschlüsseln ohne Klartext

Während ich mit der Umsetzung meiner Arbeit beschäftigt war, waren die ersten Prototypen des SMG (in ihrer ursprünglichen Form) bereits im Testeinsatz. Solange das SME nur auf SMGs aufgebaut war, stellte sich die Frage der Erzeugung von Klartext auf dem Weg zum Empfänger nicht. Mit der Komponente SMS entstand die Frage, wie man eine Umverschlüsselung realisieren kann, ohne jedoch dabei im SMS den Klartext aller E-Mails (zumindest temporär) vorliegen zu haben.

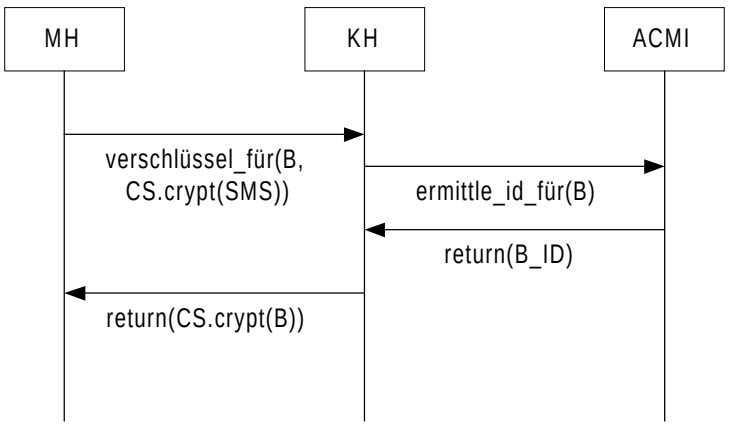
Dieses Problem löste die Arbeit von Michael Herfert der GMD Darmstadt. Michael Herfert entwikkelte die Idee des Messagehandlers und des Keyhandlers als getrennte Module [Herf97].

Wie bereits erwähnt, werden für die Verschlüsselung der E-Mails hybride Verfahren eingesetzt. Das heißt, daß die Mail mit einem symmetrischen Schlüssel verschlüsselt ist (dem Session Key) und dieser Session Key dann mit dem asymmetrischen öffentlichen Schlüssel des Empfängers verschlüsselt wird.

Gelingt es nun, den SMS in zwei Teile zu teilen, nämlich den Message Handler (MH) und den Key Handler (KH), und gelingt es ferner, dem MH den auf den privaten Schlüssel des SMS zu verweigern und dem KH den Zugriff auf die komplette Mail zu verweigern, kann wie folgt vorgegangen werden:

Der MH extrahiert aus der Mail den mit dem öffentlichen Schlüssel des SMS verschlüsselten Session Key. Diesen verschlüsselten Session Key sendet der MH zusammen mit dem gewünschten Empfänger an den KH. Der KH entschlüsselt den Session Key und verschlüsselt ihn neu mit dem öffentlichen Schlüssel des Empfängers. Dann sendet der KH den neu verschlüsselten Session Key an den MH. Der MH hängt diesen neu verschlüsselten Session Key wieder an die E-Mail und sendet diese ab.

ABBILDUNG 11. Kommunikation zwischen Mailhandler (MH), Keyhandler (KH) und ACMI



CS ist der Session Key verschlüsselt mit dem jeweiligen öffentlichen Schlüssel des Empfängers.

Auf diese Weise entsteht die gesamte Zeit kein Klartext. Nur wenn die Systemverwalter des MH und des KH eine “kriminelle Vereinigung” bilden, können sie den Klartext einer E-Mail gemeinsam erzeugen.

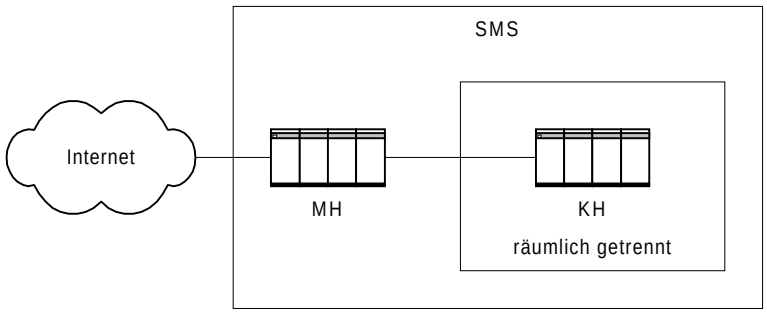
4.3.1 Trennung von MH und KH

Wie aber können MH und KH so voneinander getrennt werden, daß jeweils auf das Geheimnis des anderen nicht zugegriffen werden kann?

Die einfachste Variante wären zwei Prozesse auf einem UNIX-System, die unter unterschiedlichen Benutzerkennungen laufen. Dieser Schutz wäre sehr schwach. Der Systemadministrator dieses UNIX-Systems hätte sowohl Zugriff auf den Schlüssel des KH als auch Zugriff auf die E-Mails, die über das Internet übertragen werden.

Die zweite Möglichkeit wäre eine räumliche Trennung zwischen MH und KH. Man baut also zwei Systeme auf. Einen MH-Server und einen KH-Server. Der KH-Server kommt in einen abschließbaren Raum, der ohne weiteres nicht zugänglich ist. Ferner bekommt der KH keinen direkten Zugriff auf das Internet, sonst könnte er die E-Mails bei der Übertragung vom MH ins Netz abfangen. Es gibt lediglich eine Leitung vom MH-Server zum KH-Server, die nur dazu benutzt wird, die KH-Abfragen zu gestatten. Der KH-Administrator darf keine Daten in den Raum des KH-Server mitnehmen und keine von dort wieder heraus. Das stellt sicher, daß der Klartext der E-Mails nicht erzeugt werden kann.

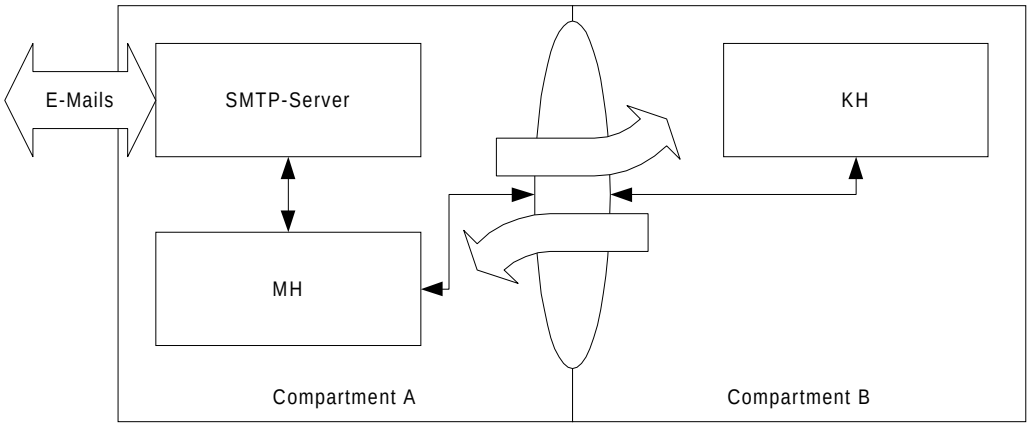
ABBILDUNG 12. Mögliche Konfiguration von MH- und KH-Servern



Die beste Möglichkeit ist die Verwendung eines sicheren Betriebssystems (engl. trusted operating system). Ein Beispiel hierfür ist das von Hewlett Packard (HP) entwickelte Compartmented Mode Workstation (CMW) System. Dieses System ist ein modifiziertes UNIX-Betriebssystem, dem der Superuser fehlt. Es gibt also nach der Installation dieses Systems keine Person mehr, die auf alle Bereiche der Maschine Zugriff hat. Die Maschine selbst implementiert so etwas wie verschiedene virtuelle Maschinen, die sich alle wie ein vollständiges System verhalten, jedoch nicht an alle Ressourcen kommen, die die anderen virtuellen Maschinen verwalten. Diese virtuellen Maschinen werden Compartments genannt. Ein Compartment kann mit einem anderen Compartment über sog. Gateways kommunizieren. Diese Gateways sind sehr kleine und damit gut verifizierbare Programme. Nur ein Compartment hat in unserer Konfiguration Zugriff auf das Internet. Man kann sich nun vorstellen, wie der MH in einem anderen Compartment läuft als der KH. Damit ist ebenfalls das Prinzip der Teilung der Verantwortlichkeit (engl.: seperation of duty) gewährleistet.

Vereinfacht dargestellt könnte eine Konfiguration unter CMW wie in der Grafik (Abbildung 13 auf Seite 35) dargestellt aussehen. Der MH läuft in einem anderen Compartment als der KH. Beide können über das Gateway miteinander kommunizieren.

ABBILDUNG 13. Vereinfachte Darstellung einer CMW-Konfiguration mit MH und KH



KAPITEL 5

Analyse

Nachdem die ersten Überlegungen zur Funktion und darauf aufbauend zur Architektur des zu entwikkelnden Systems gemacht sind und fest steht, daß das Projekt durchführbar ist, ist es nun an der Zeit, eine Software-Entwicklungsmethode auszuwählen.

Ich möchte im praktischen Teil meiner Arbeit weitgehend eine Software-Entwicklung nach der Fusion Methode durchführen. Diese Methode wird an der TU in der Basisveranstaltung “Softwaretechnik und Systemgestaltung” gelehrt und erfüllt moderne Anforderungen an eine Softwareentwicklungsmethode.

Einige Entwürfe oder Implementierungen weichen von den zunächst erarbeiteten Lösungen ab. Aus welchen Gründen jeweils welche Änderungen gegenüber der ursprünglichen Planung entstanden sind, werde ich an entsprechender Stelle erläutern. Nicht alle zunächst modellierten Klassen werden später in meinen Programmen verwendet. Aber auch auf diesen Sachverhalt werde ich an geeigneter Stelle noch einmal genau eingehen. Das Hauptproblem besteht im Grunde darin, daß in dieser Arbeit ein dynamischer Prozeß statisch abgebildet wird.

Die hier dargestellten Diagramme und Erläuterungen sind alle die Ergebnisse verschiedener Zwischenschritte. Diese alle aufzuzeigen, würde jedoch den Rahmen dieser Arbeit sprengen. Somit beschränke ich mich darauf, das Endprodukt zu beschreiben und im Text einige Anmerkungen dazu zu machen, wie es zu diesem Ergebnis gekommen ist. Ferner werde ich nicht alle Diagramme und Programmdokumentationen vollständig mit in die Arbeit aufnehmen. Die vollständige Programmdokumentation ist besser auf einem Datenträger aufgehoben und kann dort bei Bedarf als Nachschlagewerk dienen.

An dieser Stelle möchte ich noch einmal deutlich machen, daß für mich die Software-Entwicklungsmethode mit ihren Darstellungen nicht nur die Aufgabe hat, zu einer guten Implementierung zu führen. Sie stellt auch einen wichtigen Bestandteil der Programmtext-Dokumentation dar und ist somit auf dem Datenträger neben den Programmtexten gut aufgehoben. Das ist nicht besonders verwunderlich, da wir uns als Softwareentwickler üblicherweise in einem Kreis der ständigen Weiterentwicklung befinden und somit zwangsläufig die Dokumentation der alten Programmversion gleichzeitig Ausgangspunkt für die neue Entwicklung ist.

5.1 Die Anforderungsanalyse

Wie so manches Software-Projekt ist auch dieses als Modellprojekt für Software-Engineering-Methoden ungeeignet. Kein Projekt, bei dem ich bislang mitgearbeitet habe, ließ sich sauber in Analyse-, Entwurfs- und Implementierungs-Phase zergliedern, und an manchen Stellen traf nicht einmal die Idee

des Spiralmodells zu, nachdem die drei Phasen bis zum endgültigen Produkt immer wieder neu durchlaufen werden. Betrachtet man die Entwicklung des SME, läßt sich diese Spirale deutlich erkennen. Die Abweichung von einem Modellprojekt liegt in der Anfangsphase.

Eine wie auch immer geartete Anforderung führte zu neuen Entwürfen und schließlich zu Prototypen. Die Prototypen zeigten die Schwächen der erarbeiteten Modelle auf und führten zu erweiterten Anforderungen usw. Das Kapitel „Automatisierung von sicherem elektronischen Schriftverkehr“ auf Seite 25 beschreibt zum Teil die Entwicklung, die im Laufe des Projektes stattgefunden hat.

Die Fusion Methode geht davon aus, daß dem Entwicklerteam eine Anforderungsdefinition als Ergebnis einer Anforderungsanalyse vorliegt, nach der die einzelnen Modelle entwickelt werden können. Wie es zu dieser Anforderungsdefinition kommt, ist nicht Bestandteil der Methode. Die Tatsache, daß dieser Teil der Softwareentwicklung von der Fusion Methode ausgeschlossen ist, hat durchaus seine Gründe. Ohne Zweifel handelt es sich bei einer Anforderungsdefinition um eines der wichtigsten, wenn nicht das wichtigste Dokument im gesamten Entwicklungsprozeß. Als Startpunkt der Entwicklung kann es entscheidend für den Erfolg oder Mißerfolg des gesamten Projektes werden. Die Modellierung, der Entwurf und schließlich die Implementierung können gerade nur so gut sein wie die Anforderungsdefinition, da sie als Produkt aus diesen hervorgehen.

Wie bereits erwähnt, handelt es sich bei der Entwicklung des SME um kein gewöhnliches Projekt, bei dem Verantwortliche befragt oder Arbeitsplätze analysiert werden konnten, um daraus Anforderungen an das System herzuleiten. Stattdessen gibt es hier einen sehr forschungsorientierten Ansatz, der über die ständige Erweiterung von Prototypen zum Produkt kommen soll. Mit meiner Arbeit ist der letzte Schritt in diesem Kreis noch nicht erreicht. Diese Arbeit dokumentiert einen Meilenstein in dieser Entwicklung. Das SME ist so weit, daß es organisationsintern einsatzfähig ist und Ansätze für den Austausch mit externen Kommunikationspartnern bietet. Welche Schritte als nächstes folgen könnten, beschreibe ich im Kapitel: „Aussichten“ auf Seite 135.

Im Folgenden stelle ich die Anforderungen zusammen, die zur Entwicklung des SME in seiner aktuellen Version geführt haben. Wie sich zeigen wird, bieten die erarbeiteten Konzepte an einigen Stellen mehr, als das geforderte Verhalten. Ich stelle hier also die Mindestanforderung an das System zusammen.

5.1.1 Funktionale Anforderungen

- **F.1:** E-Mails, die unsichere Netzabschnitte passieren, müssen kryptographisch gesichert werden.
- **F.2:** Der Inhalt von E-Mails darf nicht abstreitbar sein (digitale Unterschrift).
- **F.3:** Das System darf keine zusätzlichen Qualifikationen von den Mitarbeiterinnen und Mitarbeitern fordern.
- **F.4:** Es muß erkennbar sein, welche E-Mail als sicher eingestuft werden kann und welche unsigniert und unsicher das System erreicht hat.
- **F.5:** Das System darf keine Bindung an ein bestimmtes E-Mail Produkt darstellen (keine Erweiterung von bestehenden Programmen), da sonst eine finanzielle Bindung an den Hersteller besteht.
- **F.6:** Es darf keine organisatorische Bindung an eine Organisation bestehen, die beispielsweise die Schlüsselverwaltung übernimmt.
- **F.7:** Personen müssen über ihr Stellenzeichen indirekt und beispielsweise über ihre Personalnummer direkt ansprechbar sein.
- **F.8:** Das System sollte Rundschreiben ermöglichen (z.B. “an alle studentischen Hilfskräfte”, “an alle Professoren”, “an alle Projektleiter”, “an die MitarbeiterInnen des Instituts X”).
- **F.9:** Das System sollte dem Signaturgesetz entsprechend korrekt arbeiten.

5.1.2 Nichtfunktionale Anforderungen

- **N.1:** Das System sollte sowohl unter UNIX-Umgebungen als auch unter Windows lauffähig sein.
- **N.2:** Die Anbindung von weniger verbreiteten Betriebssystemen sollte in jedem Fall möglich sein.
- **N.3:** Das System sollte nicht auf eine Verschlüsselungsmethode oder eine Schlüssellänge beschränkt sein, da davon ausgegangen werden muß, daß sich die Anforderungen hier schnell ändern und eine Anpassung möglich sein muß.
- **N.4:** Da Smartcards dem heutigem Stand der Technik entsprechen, sollten diese als Träger eines privaten Schlüssels benutzt werden können.

5.1.3 Architekturbedingte Anforderungen

- **A.1:** Bei der Weiterleitung von E-Mails im zentralen Mailserver darf kein Klartext erzeugt werden.
- **A.2:** Ein Empfang einer Nachricht durch einen unbefugten Kommunikationsparter soll dem heutigem Stand der Technik entsprechend ausgeschlossen werden.

5.2 SME - Architektur

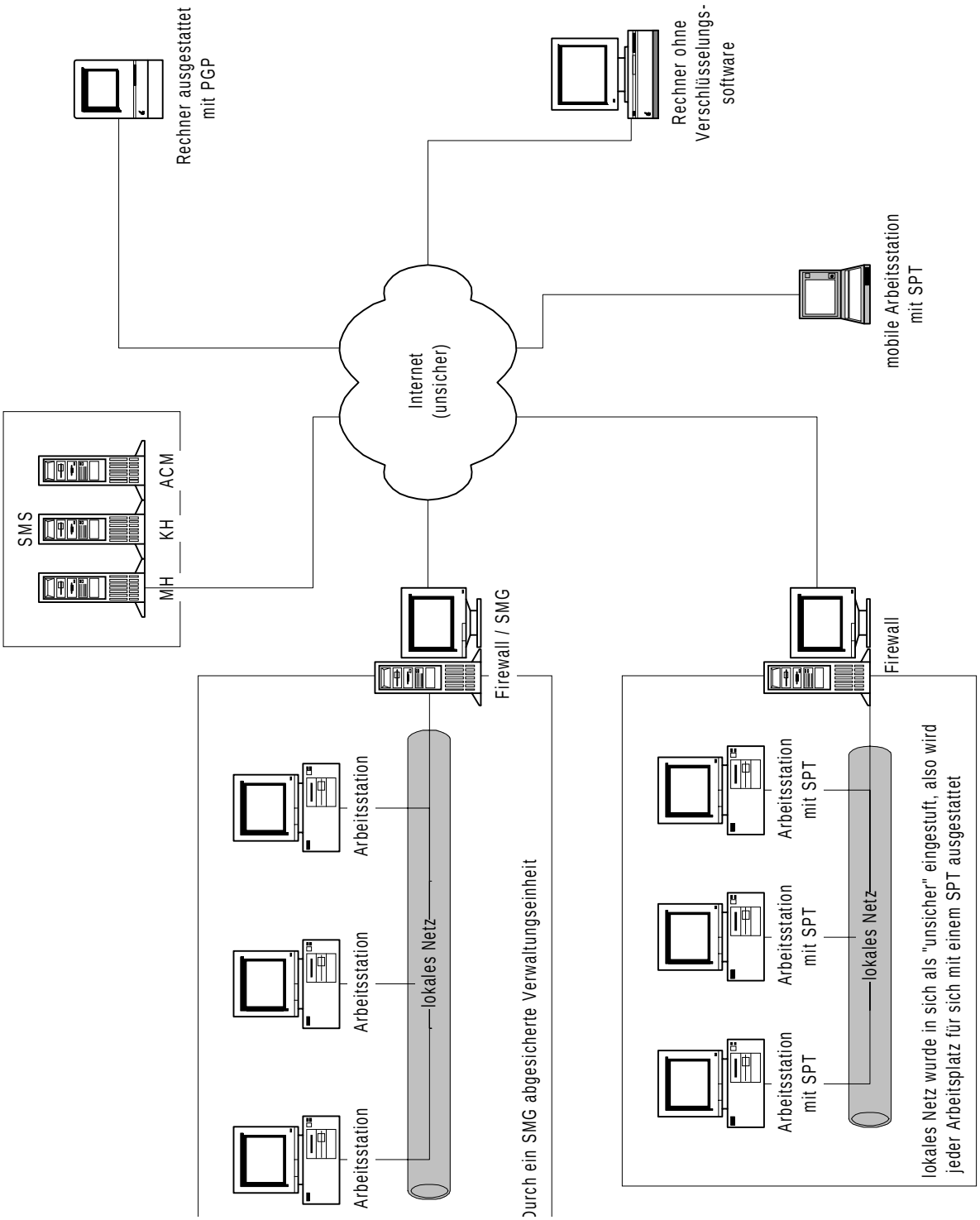
Ich möchte an dieser Stelle zusammenfassen, wie die Softwarearchitektur eines SME aussieht. Ich stelle diese Architekturüberlegungen der Fusion Analyse Phase voran. Dies ist notwendig, weil sich aus der Architektur einer Software fast immer zwangsweise weitere Anforderungen ergeben, die im weiteren Entwicklungsprozeß berücksichtigt werden müssen.

Ein SME kann, wie bereits dargelegt, folgende Komponenten enthalten:

- Kernstück bildet der Secure Mail Server (SMS), der aus zwei bis drei Rechnern bestehen oder auf verschiedenen Compartments einer CMW Workstation laufen kann.
- Ein durch einen Secure Mail Gateway / Firewall (SMG) abgesichertes Teilnetz (Intranet).
- PCs, die mit den Secure Proxy Tools ausgestattet sind.
- Eingeschränkt ist auch ein nicht sicherer oder nur zum Teil abgesicherter E-Mail Verkehr mit Rechnern mit Standardverschlüsselungssoftware (PGP, PEM, S/MIME) bzw. Rechner ohne Encryptionsoftware möglich.

Die folgende Grafik veranschaulicht die Vernetzung der einzelnen Komponenten.

ABBILDUNG 14. Komponenten des SME



Bevor wir uns nun der Analyse der einzelnen Komponenten widmen können, müssen wir uns zunächst über die Softwarearchitektur klar werden. Wir kennen die grobe Struktur der SME. Widmen wir uns nun der Struktur der einzelnen Komponenten:

5.2.1 Architektur des SMG

Der SMG soll in der Lage sein, Mails zu empfangen, zu verschlüsseln und zu versenden, bzw. zu empfangen, zu entschlüsseln und zu versenden. Der Vorgang des Empfangens und des Versendens ist nicht ganz trivial. Die Geschichte des Programms Sendmail, welches zur Zeit eines der am häufigsten gebrauchten SMTP-Server ist, zeigt, wie schwierig es ist, ein solches Programm zu entwickeln, ohne daß es irgendwelche Sicherheitslücken gibt, die von Crackern benutzt werden können. Ich stand also vor der Frage, wie ich selbst ein solches Programm umsetzen sollte. Stattdessen verfolgte ich einen anderen Weg:

Ich entschied mich den SMG als Filter zu entwerfen (Architekturmuster Pipes-and-Filters [Busch98]). Das SMG-Programm implementiert selbst keine Sende- oder Empfangsfunktionen, sondern überläßt diesen Teil der Funktionalität einem SMTP-Server. Auf der einen Seite kann das Konzept auf diese Weise nicht sicherer sein, als der SMTP-Server selbst, ich gehe jedoch davon aus, daß ein Programm wie Sendmail immer auf dem neuesten Stand der Entwicklung gehalten wird und der SMG-Filter davon unberührt bleiben kann. So spielt an dieser Stelle eigentlich nicht einmal das Übertragungsprotokoll SMTP eine Rolle. Das heißt, daß so die Übertragungskomponente austauschbar ist. Die Umsetzung sieht vor, daß der SMTP-Server beim Eingehen einer E-Mail den SMG startet und diesem die E-Mail zur Ver- bzw. Entschlüsselung übergibt. Der SMG selbst verrichtet seine Arbeit und startet wiederum ein externes Programm zum Versenden der Nachricht.

ABBILDUNG 15. Filterfunktion des SMG

Eine Unterscheidung muß hier für das Empfangen von E-Mails und das Versenden von E-Mails aus Sicht des Netzes hinter dem SMG-Rechner gemacht werden. Es muß also unterschieden werden, ob eine E-Mail vom lokalen Netz ins Internet geht oder ob die Mail vom Internet kommt und ins lokale Netz geleitet werden soll. Ich führe daher zwei Filter ein: SMG-Send für Mails vom lokalen Netz in das Internet und SMG für den Empfang von Mails aus dem Internet.

SMG-Send ist also für das Verschlüsseln von E-Mails zuständig, SMG für das Überprüfen und Entschlüsseln.

ABBILDUNG 16. Filterfunktion des SMG-Send

Die Abbildungen zeigen den Weg der E-Mail durch das System. Dabei stehen die Kästen jeweils für ein Filtermodul, die Pfeile symbolisieren die Pipes. In den Pfeilen steht jeweils die Typisierung der Daten, die von Filter zu Filter übertragen werden. Dabei wäre noch genauer zu spezifizieren, was eine “verschlüsselte E-Mail” oder eine “uverschlüsselte E-Mail” ist.

5.2.2 Architektur des SMS

Warum sollten wir nicht die gleiche Architektur des SMG auch für den SMS anwenden? Hier gestaltet sich die Arbeit sogar noch einfacher, da hier keine zwei Fälle unterschieden werden müssen. Der Vorgang ist immer der gleiche:

Sendmail empfängt eine E-Mail und leitet sie an den SMS-Filter weiter. Der SMS-Filter übernimmt die Überprüfung der E-Mail, löst die erweiterten E-Mail Adressen auf (mittels ACM) und initiiert die Umverschlüsselung. Das Umverschlüsseln und Absenden der Nachricht wird ggf. für jeden Empfänger dieser E-Mail wiederholt.

Ein Beispiel: Es wird eine E-Mail an die Adresse “E2S” geschickt. Hinter dieser Adresse verbergen sich 6 Personen, die dem Projekt angehören. Im ACM ist das dadurch modelliert, daß die 6 Personen Mitglieder beim Objekt “E2S” sind. Der SMS wird für jede der 6 Personen eine Umverschlüsselung der E-Mail vornehmen und nach erfolgreicher Umverschlüsselung nacheinander 6 Sendmail-Prozesse starten, die für die Auslieferung der E-Mail verantwortlich sind.

5.2.3 Architektur der SPT

Wie schon diskutiert, werden für alleinstehende PCs oder für solche, die erhöhten Sicherheitsanforderungen gerecht werden sollen, Proxies erstellt, die zwischen POP3-Server und Mailclient bzw. SMTP-Server und Mailclient geschaltet werden. Diese Proxies übernehmen dann im Grunde die Aufgabe des SMG. Unterschied ist, daß sie nicht als Filter, sondern als Proxy implementiert sind.

5.2.4 Die Rolle des POP3 Daemons

Das SMTP-Protokoll ist so konzipiert, daß E-Mails über eine Internet Socketverbindung von einem beliebigen Rechner an den SMTP-Server gesendet werden können. Heute wird der Zugriff der Rechner auf den SMTP-Server oft eingeschränkt, um den Mißbrauch durch sog. Spam-Mails entgegenzuwirken [Cost97].

Das SMTP-Protokoll sieht jedoch kein Abfragen von E-Mails über das Netzwerk vor. Die Idee bei der Erstellung des Protokolls war offensichtlich die, daß der Empfang von E-Mails die Aufgabe eines zentralen Rechners ist, an dem sich die Benutzer dann über ein Terminal anmelden können, um ihre E-Mails zu lesen.

Der nächste Schritt bestand darin, die Verzeichnisse mit den empfangenen E-Mails an UNIX-Workstations über das Netzwerkprotokoll Network File System (NFS) zu exportieren, so daß von den Workstations aus die Illusion entsteht, die eigene Workstation hätte die Mails empfangen und auf einer lokalen Festplatte gespeichert. Das Versenden wurde über entsprechende Konfigurationen realisiert.

Im Zeitalter der Windows PCs und Network Computer (NC), wie auch Rechner, die sich über Telefon oder Datenleitungen in die Netze einwählen, um u.a. auch E-Mails auszutauschen, benötigt man andere Konzepte. Dieses Konzept wurde durch POP3 und IMAP realisiert. In meiner Arbeit beschränke ich mich auf das POP3 Protokoll. Lösungen für IMAP sehen ähnlich aus, bedürfen jedoch einer ganz eigenen Diskussion [Krol95].

POP3 ist eine Art Postfach-Dienst, der E-Mails aufhebt und auf Anfrage über eine Internet Socketverbindung bereitstellt. Damit kann der Benutzer bequem von zu Hause aus seine E-Mails vom Mailserver seiner Dienststelle oder vom Mailserver seines Internet Service Providers abrufen. Hierbei werden in der Regel alle E-Mails zum PC übertragen, so daß die Verbindung danach wieder abgebaut werden

und der Benutzer in aller Ruhe seine E-Mails lesen kann, ohne daß ihm weitere Telefonkosten entstehen.

In der Architektur und in den Systemobjektdiagrammen taucht der POP3 Daemon (in der UNIX - Welt werden Serverprogramme auch oft Daemon genannt, da diese im Hintergrund ihre Arbeit verrichten und nicht direkt mit dem Benutzer in Kontakt treten und man nur durch ihren Dienst bemerkt, daß sie da sind [Lef90]) nicht auf. Der Grund dafür ist ganz einfach. Der SMTP-Server ist dafür verantwortlich, die E-Mails zu empfangen und auf der Festplatte zu speichern. Der POP3-Server ist lediglich dafür zuständig, die vom SMTP-Server abgelegten E-Mail Daten an die Benutzer weiterzuleiten.

Interessant wird der POP3-Daemon noch einmal bei der Diskussion über die Sicherheit des Systems, da er mit seinem Protokoll und durch sein bloßes Vorhandensein selbstverständlich einen Angriffspunkt bietet. Auch ist interessant, daß wir den POP3-Daemon benutzen, um verschlüsselte und unverschlüsselte E-Mails zu übertragen.

Wird die E-Mail auf einem SMG in einem als sicher angenommenen lokalen Netzwerk abgelegt, so geschieht das hinter dem SMG Filter unverschlüsselt, siehe „Filterfunktion des SMG“ auf Seite 41.

Wird die E-Mail auf einem SMS an einer zentralen Stelle abgelegt, befindet sie sich also im als unsicher eingestuften Netzabschnitt, so wird sie verschlüsselt abgelegt. Das wird z.B. vom SMS selbst so realisiert. Die E-Mails für SPT Benutzer, also solchen mit PCs, die sich nicht hinter einem SMG befinden, werden direkt verschlüsselt in die entsprechenden E-Mail Verzeichnisse einsortiert. Die Entschlüsselung übernehmen in diesem Fall die SPTs.

5.3 Objektmodelle

Die Fusion Methode sieht vor, daß im ersten Schritt der Analysephase ein sogenanntes Objektmodell für das zu entwickelnde System erstellt wird. Dieses Objektmodell stellt alle beteiligten Objekte und deren Beziehungen untereinander dar. Es dient also, so könnte man sagen, der Verdeutlichung, wer die Akteure sind und wie sie zueinander stehen. Ausgehend von diesem Objektmodell werden dann weitere Analysen durchgeführt.

Die Fusion Methode ist, wie der Name schon sagt, eine Kombination aus verschiedenen Methoden. Zur Darstellung der Objektmodelle verwendet man hier ein erweitertes Entity-Relationship Diagramm. Entities werden hier mit Objekten gleichgesetzt, die Relationships sind die Beziehungen unter den Entities.

Wärend das Objektmodell sowohl diejenigen Objekte erfaßt, die später im System implementiert werden, als auch die Objekte, die von außen auf das System einwirken, wird im zweiten Schritt die Grenze zwischen dem System und der Außenwelt gezogen. Dies wird im Objektmodell durch eine gestrichelte Linie dargestellt. Alle Objekte außerhalb der gestrichelten Linie sind sog. Agenten, die auf das System in irgendeiner Form einwirken oder mit ihm in irgendeiner Beziehung stehen. Alle Objekte innerhalb der gestrichelten Linie sind Bestandteile des Systems und müssen in irgendeiner Form realisiert werden.

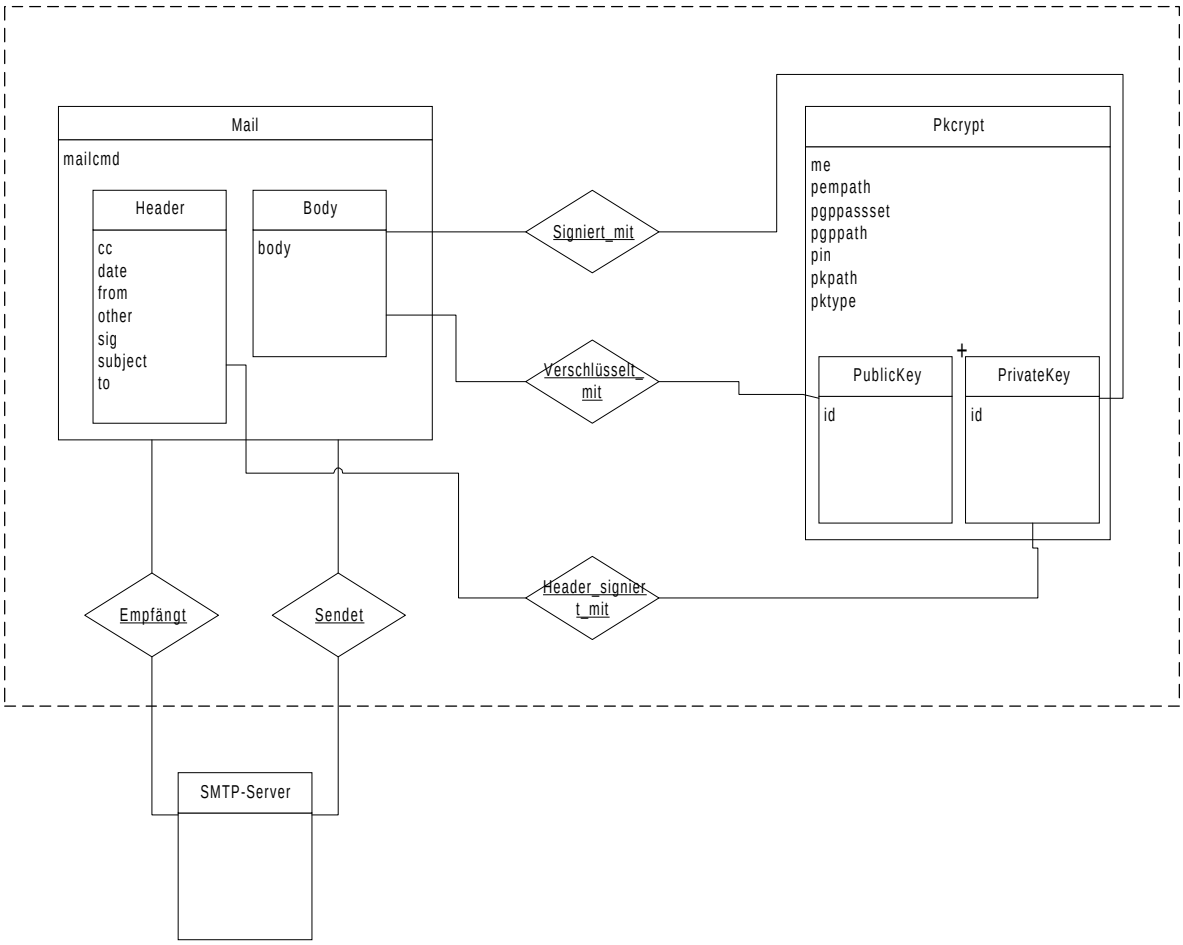
Ich möchte diese Entwicklung nun exemplarisch an der Modellierung des SMG und des SMS zeigen.

5.3.1 Secure Mail Gateway (SMG)

Die folgende Abbildung zeigt das Systemobjektmodell des SMG. Das Modell würde für den SMG-Filter und den SMS-Send-Filter genauso aussehen, da jeweils die gleichen Objekte beteiligt sind. Der Unterschied besteht hier im Grunde nur im Ablauf des Programms, also wie das Programm die Objekte benutzt.

Ich möchte nun kurz erläutern, auf welchen Grundlagen ich dieses Modell erstellt habe. In den beschreibenden Texten markiere ich die Objekte, die im Modell wiederzufinden sind, die Relationen ergeben sich aus den Zusammenhängen.

ABBILDUNG 17. Systemobjektmodell: SMG



Der SMG empfängt die Nachrichten (**Mail**) vom SMS mittels **SMTP-Server**. Eine **Mail** besteht aus einem **Header** und einem **Body**. Der SMG überprüft die Signatur von Header und Body mit dem öffentlichen Schlüssel des SMS (**PublicKey**) und entschlüsselt den Body mit seinem eigenen privaten Schlüssel (**PrivateKey**). Danach wird die geprüfte, entschlüsselte Mail wieder an den SMTP-Server übergeben. Für die Prototypen benutzen wir jeweils das Programm Sendmail als SMTP-Server. Die Funktionen Verschlüsselung, Entschlüsselung, Signierung und Prüfung der Signatur übernimmt ein Public-Key-Kryptoverfahren (**Pkcrypt**). PrivateKey und PublicKey sind Bestandteile des Public-Key-Kryptoverfahrens. Sie tauchen im Diagramm als Bestandteil von Pkcrypt auf.

Wie schon erwähnt, benutzt der SMG-Send-Filter genau die gleichen Objekte, entspricht bei genauer Betrachtung also dem gleichen Systemobjektmodell, wie der SMG-Filter.

Das SMG-Send Programm empfängt aus dem lokalen Netz die **Mail** mittels **SMTP-Server**, verschlüsselt diese mit dem öffentlichen Schlüssel des SMS (**PublicKey**) und signiert den Nachrichtentext (**Body**) sowie die normierten Headerzeilen (**Header**) mit dem privaten Schlüssel (**PrivateKey**). Zu beachten ist dabei, daß nicht der gesamte Header unterschrieben wird, sondern nur ein Teil. Diesen modelliere ich jedoch nicht getrennt. Er kann als Teilmenge des Headers jederzeit als dem Header gebildet werden. Selbstverständlich arbeitet auch das SMG-Send mit dem Public-Key-Kryptoverfahren (**Pkcrypt**).

Aus Modellierungssicht befindet sich der SMTP-Server außerhalb der zu implementierenden Klassen. Es handelt sich hierbei um ein Objekt (genauer eine Klasse von Objekten), das von außen auf unsere Objekte einwirkt (“Agent” genannt). Genauer: Der SMTP-Server löst von außen die Funktionskette des zu entwickelnden Programms aus. Man spricht hier von Ereignissen. Er empfängt E-Mails und leitet diese an die entsprechenden Filter weiter. Die Filter selbst rufen dann den SMTP-Server wiederum auf, um die gefilterten Daten auszuliefern. Wie nicht anders zu erwarten, offenbart das Systemobjektmodell, das die Klasse “Mail” den zentralen Punkt im System darstellt und offensichtlich die Klasse “Pkcrypt” eine weitere wichtige Rolle spielt.

Die Klasse Mail soll dabei in meinem Modell eine ARPA-Message nach RFC 822 mit den von mir entworfenen Erweiterungen (Headersignatur) darstellen. Pkcrypt ist ein sehr abstraktes Modell einer Public-Key-Kryptographie. In dieser Klasse verbergen sich öffentliche und private Schlüssel sowie andere Attribute, die zur Benutzung verschiedener Public-Key Verfahren notwendig sind.

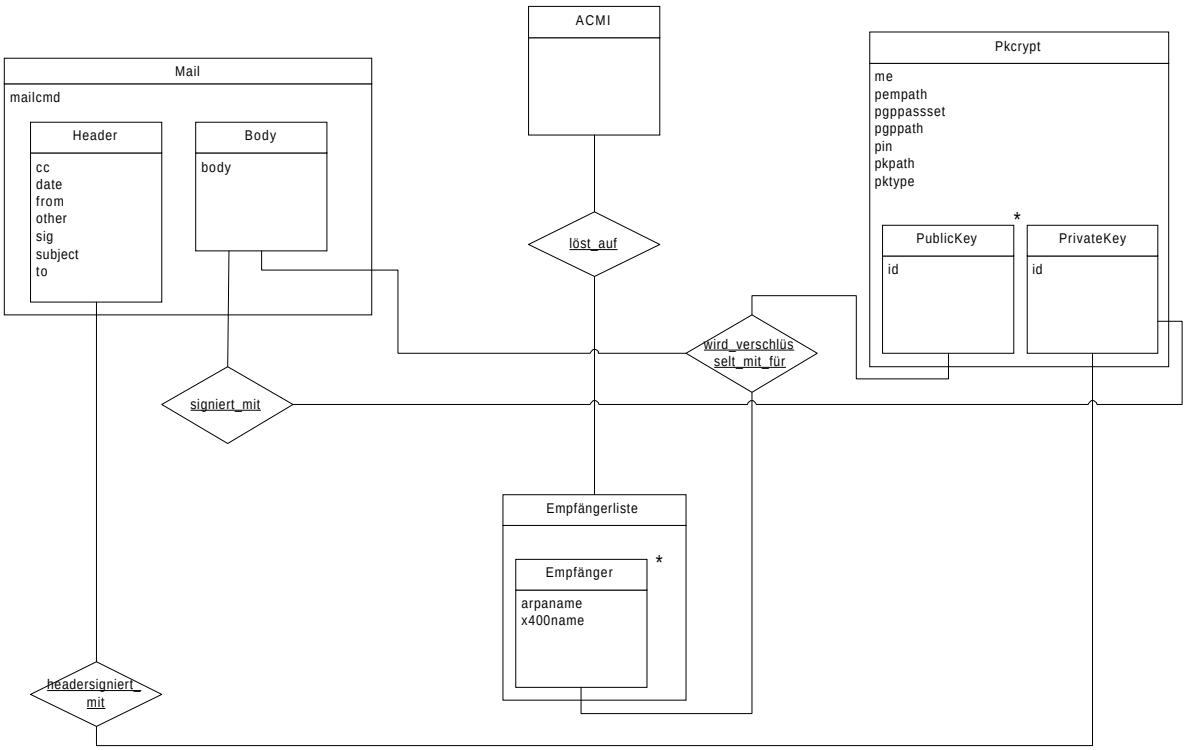
5.3.2 Secure Mail Server (SMS)

Auch für den SMS möchte ich ein Objektmodell entwerfen. Dabei wird auffallen, daß ich einige Objekte benutze, die ich schon für das SMG Objektmodell modelliert habe. Das erneute Auftauchen dieser Objekte hier im SMS Objektmodell zeigt, daß es sich offenbar um Objekte handelt, die man einmal implementieren und dann für das andere Programm wiederverwenden kann.

Die Aufgabe des Secure Mail Servers ist es,

- verschlüsselte Mails zu empfangen, sie für den Empfänger umzuverschlüsseln, ohne dabei Klartext zu erzeugen,
- Rundschreiben aufzulösen und eine Verschlüsselung für eine Reihe von Empfängern vorzunehmen,
- im Krankheits-, Urlaubsfall oder aus anderen Gründen Mails an Stellvertreter bzw. Stellennachfolger oder Bevollmächtigte umzuleiten
- und einen E-Mailverkehr an Teilnehmer weiterzuleiten, die Standardverschlüsselungsoftware benutzen.

ABBILDUNG 18. Objektmodell: SMS



Zur Durchführung benutzt der Secure Mail Server zwei externe Server: Den ACM Interpreter (**ACMI**), der Stellen, Mailinglisten u.ä. in E-Mailadressen auflösen kann (**Empfängerlisten**) und Regeln über Stellvertreter, Urlaubsvertretung usw. berücksichtigt und den Keyhandler (hier dargestellt durch das Objekt **Pkcrypt**), der die eigentliche Umverschlüsselung tätigt. Wie der Keyhandler in das Objektmodell paßt, beschreibe ich in Kapitel „Der Keyhandler“ auf Seite 88.

Wie bereits erwähnt, muß das SME zu einem Benutzer mindestens zwei Adressen kennen: Zum einen die Internet E-Mailadresse und zum anderen den Schlüssel-Identifizierer.

5.3.3 Die Secure Proxy Tools (SPT)

Das Objektmodell der SPT sieht genau so aus, wie das des SMG bzw. des SMG-Send. Hier finden exakt die gleichen Abläufe mit genau den gleichen Objekten statt und somit haben die Objekte auch die gleichen Relationen zueinander.

Die einzige Änderung sieht man im Systemobjektmodell. Die empfangene E-Mail wird hier nicht von einem SMTP-Server an den Filterprozeß weitergeleitet, der Filterprozeß ist hier über einen Proxy realisiert (Architekturmuster Proxy [Busch98]).

5.4 Timeline Diagramme

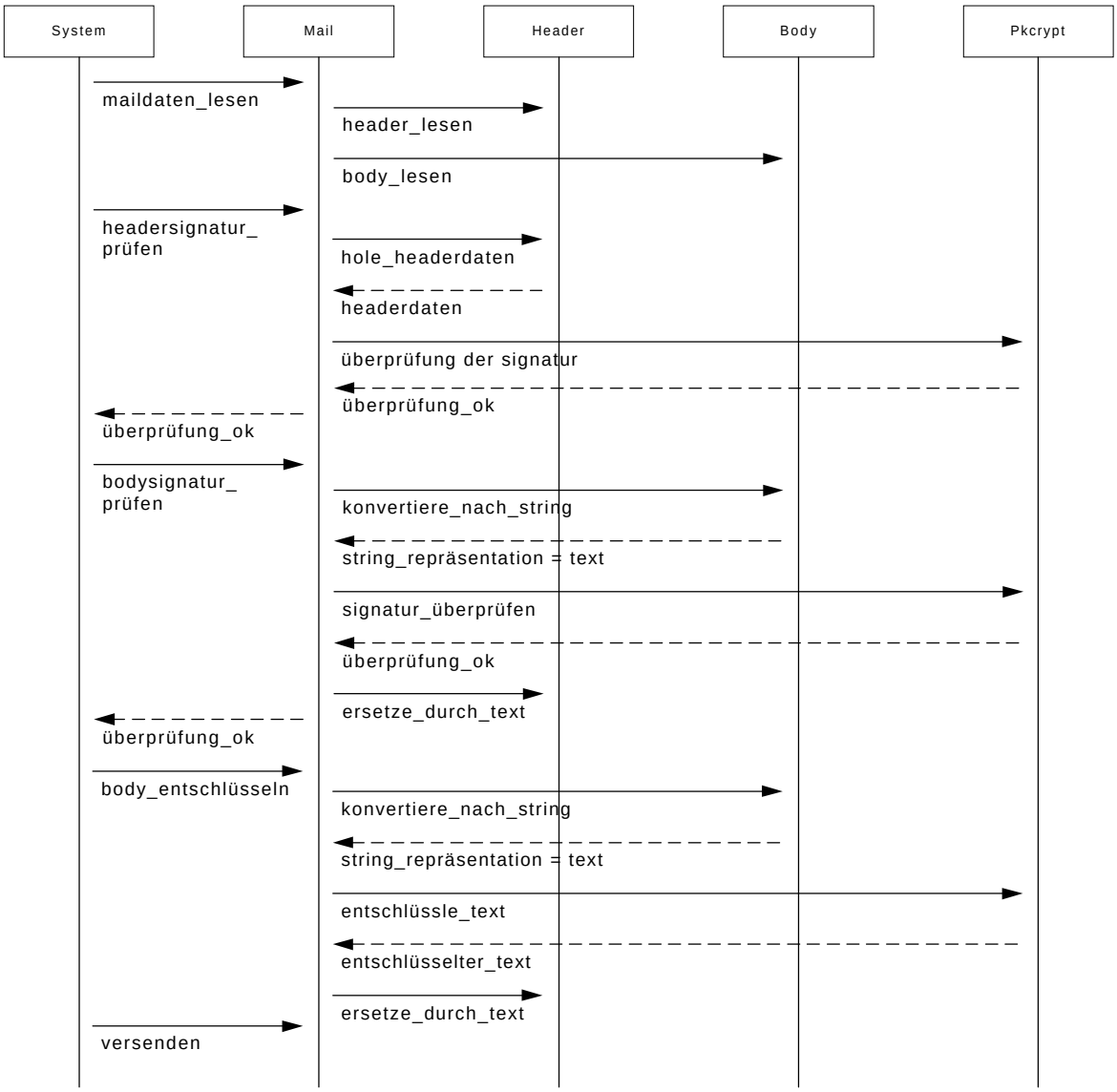
Timeline Diagramme dienen der Visualisierung von Szenarien. Die Idee hinter der Erstellung von Szenarien ist die, sich zunächst über Abläufe und Schnittstellen der Objekte untereinander und der Systemgrenzen klar zu werden. Ziel der Szenarien ist es nicht, alle möglichen Abläufe in den Programmen darzustellen. Es geht bei diesem Entwicklungsschritt darum, anhand von Beispielen ein Gefühl für das System- und Objektverhalten zu bekommen um daraus Informationen über die benötigten Operationen zu bekommen, die auf den einzelnen Objekten ausgeführt werden können müssen. Mit anderen Worten: An Hand von typischen Abläufen versuche ich mir zu verdeutlichen, was ich mit den Objekten alles anfangen können möchte.

Betrachten wir nun die Kommunikation der Objekte untereinander. Bei der objektorientierten Betrachtung spricht man von Systemoperationen und Ereignissen und davon, daß Nachrichten zwischen den Objekten verschickt werden. Die folgenden Timeline-Diagramme zeigen den zeitlichen Ablauf einer solchen Kommunikation zwischen den Objekten. Die Diagramme werden von oben nach unten gelesen. Die Pfeile stellen jeweils Nachrichten dar, die zwischen den Objekten verschickt werden. Verwirrend wird an dieser Stelle die doppelte Bedeutung von Nachricht. Wenn ich im folgenden von Nachrichten spreche, meine ich Nachrichten, die ein Objekt an ein anderes verschickt. Nachrichten im Sinne von E-Mail-Nachrichten bezeichne ich als E-Mails.

5.4.1 Empfangen einer E-Mail durch SMG

Betrachten wir nun die Kommunikation der Objekte beim Empfang einer E-Mail durch den SMG. Dabei gehe ich in meinem Szenario davon aus, daß die E-Mail korrekt ist, also kein Fehler zu behandeln ist und die unverschlüsselte E-Mail ungehindert ausgeliefert werden kann:

ABBILDUNG 19. Timeline: SMG



Wie in diesem Diagramm zu erkennen ist, habe ich die Systemgrenze so gewählt, daß nicht nur der SMTP-Server, sondern das gesamte Filterprogramm außerhalb des zu implementierenden Systems liegt. Das in der Grafik dargestellte Objekt “System” repräsentiert alle Agenten, die Nachrichten von außen in das System verschicken.

Der Filter selbst wird vom SMTP-Server gestartet und startet die Operation maildaten_lesen des Mail-Objektes. Das Mail-Objekt selbst weist das Objekt Header und das Objekt Body an, die Daten zu lesen. Der Filter ruft dann die Funktion headersignatur_prüfen auf, die im Objekt Mail implementiert ist, worauf das Objekt Mail die erforderlichen Headerdaten vom Objekt Header anfordert. Mit Hilfe dieser Headerdaten kann das Objekt Mail die normalisierte Headerform erzeugen und die Überprüfung an das Objekt Pkcrypt weitergeben.

Analog funktioniert auch die Prüfung des Mailbodies. Ausgelöst vom Filter werden die Body-Daten vom Body-Objekt angefordert und zur Signaturprüfung an das Objekt Pkcrypt übergeben. Im Gegensatz zur Headerprüfung wird der signierte Headertext im Anschluß an die Überprüfung durch eine unsignierte Version ersetzt.

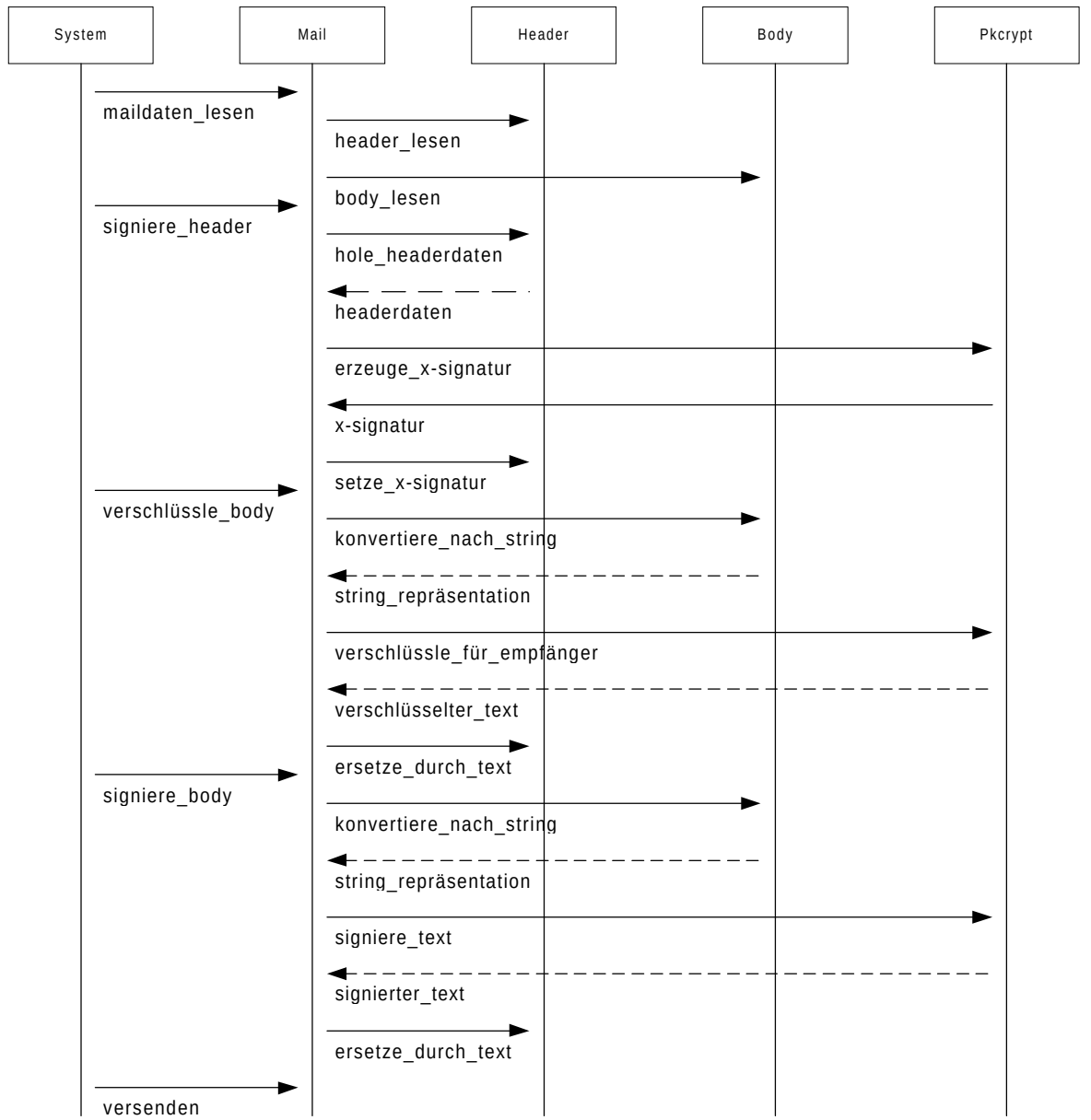
Zur Entschlüsselung der E-Mail wird der Bodytext (nun die unsignierte aber verschlüsselte Variante) wieder ausgelesen und zur Entschlüsselung an Pkcrypt übergeben. Das Ergebnis der Entschlüsselung wird dann den Bodytext ersetzen, so daß der Bodytext jetzt aus einem unsignierten, unverschlüsselten Text besteht, der versendet werden kann. Es erscheint mir logisch, daß das Objekt Mail selbst eine Operation “versenden” zur Verfügung stellt. Der Befehl müßte etwa so gelesen werden: “Mail, versende Dich!”

Werden beim Überprüfen oder Entschlüsseln Fehler erkannt, wird der Prozeß selbstverständlich abgebrochen bzw. die Mail mit entsprechenden Kommentaren weitergeleitet oder zurückgeschickt.

5.4.2 Versenden einer E-Mail durch SMG-Send

Beim Versenden der E-Mail im SMG-Send findet der umgekehrte Prozeß statt:

ABBILDUNG 20. Timeline: SMG-Send

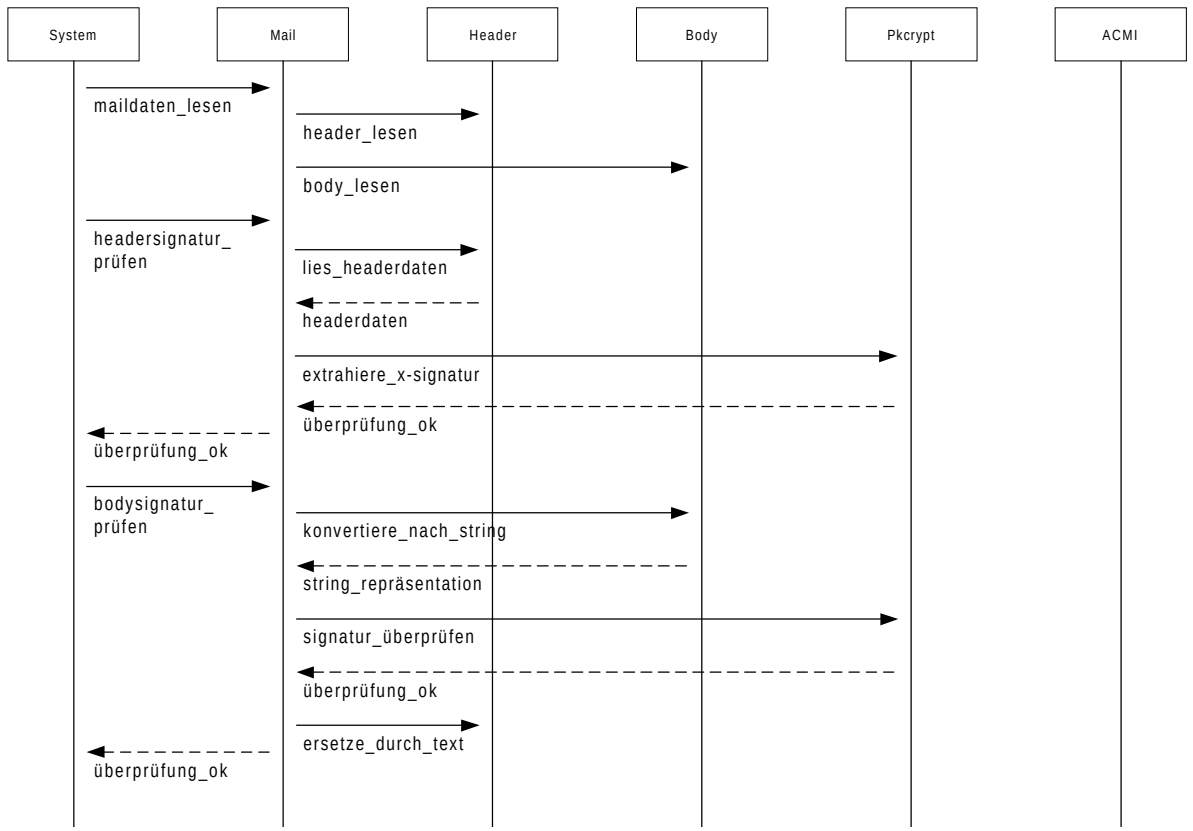


Bei der Betrachtung des SMG-Send Ablaufes gibt es keine weiteren Besonderheiten. Das Signieren des Headers findet hier in “Pkcrypt” statt. Wir finden hier die Umkehrung des Ablaufs, wie in SMG dargestellt. Nach dem Einlesen der Mail wird die X-Signatur erzeugt. Der Body wird verschlüsselt (wieder wird jeweils der Bodytext durch die verschlüsselte Variante ersetzt), dann wird der verschlüsselte Body signiert. Und an letzter Stelle steht das Versenden der E-Mail.

5.4.3 Empfangen, verarbeiten und weiterleiten im SMS

Die nun folgenden Abbildungen zeigen einen Musterablauf im SMS. Ich habe das Diagramm in zwei Teile geteilt, weil es sonst zu groß für eine Seite geworden wäre. Außerdem lassen sich die Schritte so auch übersichtlicher erläutern:

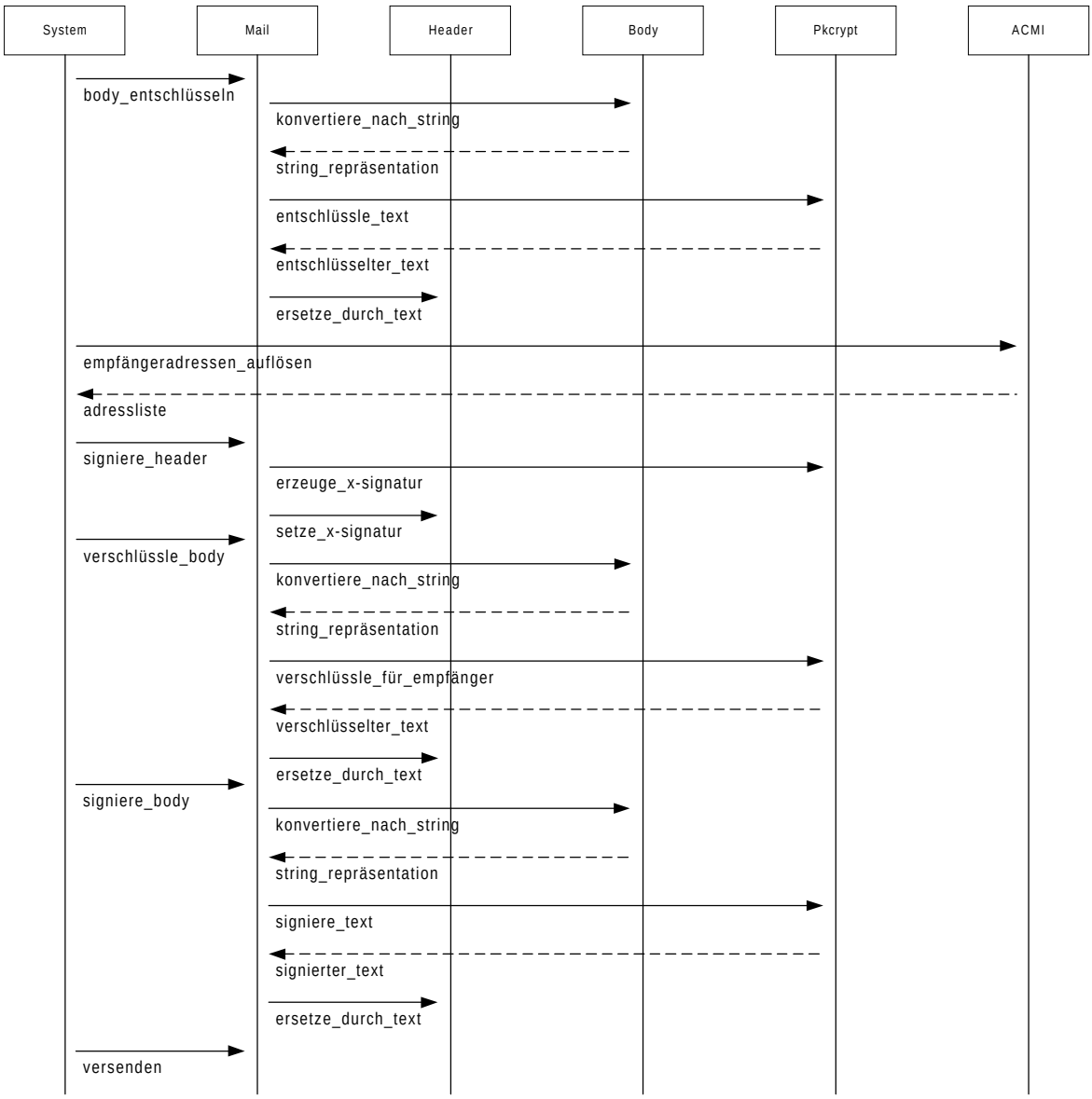
ABBILDUNG 21. Timeline: SMS (Teil 1)



Neben den aus den SMG Timeline-Diagrammen bekannten Objekten, kommt beim SMS das Objekt ACMI hinzu (vergl. Abbildung 18 auf Seite 46). Nur aus Platzgründen habe ich das Objekt Empfängerliste hier nicht dargestellt.

Wie man auf diesem ersten Abschnitt des Timeline-Diagramms erkennen kann, verläuft der erste Abschnitt im SMS identisch mit dem Vorgehen im SMG. Das ist besonders interessant, da dies bestätigt, daß wir die Objekte “Mail”, “Header”, “Body” und “Pkcrypt” nur einmal für alle Komponenten entwerfen müssen. Nicht nur die Datenrepräsentationen sind für alle Komponenten relevant, sondern auch die Operationen decken sich, wie diese Diagramme zeigen.

ABBILDUNG 22. Timeline: SMS (Teil 2)



Nachdem der Text nun entschlüsselt ist, werden die Empfängeradressen aufgelöst. Das heißt, daß der Filter eine Nachricht an das Objekt ACMI versenden wird. Diese Nachricht beinhaltet die Aufforderung zur Auflösung der logischen Mailadresse in die tatsächlichen Adressen. Dies geschieht über die Kommunikation mit dem Access Control Model Interpreter. Diese Kommunikation kapsle ich von meinem SMS System ab, indem ich ein “ACMI” Objekt modelliere, das die eigentliche Kommunikation übernimmt, den anderen Objekten jedoch einfache Operationsaufrufe zur Verfügung stellt. Das Ergebnis wird eine Liste mit Adressen und Schlüsselidentifizierern sein. Für jeden Empfänger wird dann der schon bekannte Signierungs- und Versendevorgang ausgelöst (hier etwas weiter verkürzt dargstellt).

Wie, so könnte man sich an dieser Stelle fragen, wird denn die klartextlose Umverschlüsselung gewährleistet (Siehe „Trennung von MH und KH“ auf Seite 34.)? An dieser Stelle ist noch völlig irrelevant, wie die Entschlüsselung oder die Umverschlüsselung der Mail aussehen soll. Ob der E-Mail Text an dieser Stelle wirklich unverschlüsselt vorliegt oder in Wirklichkeit nur ein Keyhandler mit

entsprechenden Daten versorgt wird, spielt an dieser Stelle noch keine Rolle. Wie die Umsetzung des Keyhandler-Konzepts später realisiert werden kann, beschreibe ich ab Seite 88.

5.5 Ablaufmodelle

Der tatsächliche Ablauf eines Programms wird in der Fusion Methode über sog. Lifecycle Diagramme modelliert. Dabei werden nicht, wie z.B. bei Flußdiagrammen üblich die einzelnen Operationsschritte über Entscheidungspunkte miteinander verknüpft. Das Lifecyclemodell gibt vielmehr Auskunft darüber, in welcher Kombination Operationen auftreten können.

Dabei werden strikt aufeinander folgende Operationen durch Punkte getrennt dargestellt. So bedeutet maildaten_lesen.headersignatur_prüfen, daß auf das Lesen der Maildaten immer die Signaturprüfung folgt. Mit einem Doppelkreuz gekennzeichnete Operationen stellen Ausgabeereignisse dar. #Fehlerinformation bedeutet also die Ausgabe einer Fehlerinformation. Der senkrechte Strich signalisiert eine Alternative. Das heißt, daß (#Bodyfehler|#BodyOk) bedeutet, daß entweder ein Bodyfehler ausgegeben wird oder ein BodyOk. Welche Bedingung zu welcher Ausgabe führt, zeigt dieses Diagramm nicht. Klammerungen, sind wie in der Mathematik, als Zusammenfassung zu verstehen, so bedeutet A.(B.C)|D, daß entweder A gefolgt von B und C ausgeführt wird oder A gefolgt von D.

5.5.1 Ablaufmodell des SMG

ABBILDUNG 23. Lifecyclemodell: SMG

```
liefecycle SMG: (Empfangen)

Empfangen      =      maildaten_lesen.headersignatur_prüfen.(
                      #Fehlerinformation|#SignaturOk).bodysignatur_prüfen.(
                      #Bodyfehler|#BodyOk).body_entschlüsseln.
                      #entschlüsselterText
```

Im Ablaufmodell finden sich die Ereignisse aus den Szenarien wieder. Die Zusatzinformation, die ich aus dem Ablaufmodell ziehen kann, sind alle möglichen Abläufe, die entstehen könnten. Wie man erkennen kann, sind z.B. folgende Abläufe denkbar:

```
maildaten_lesen.headersignatur_prüfen.#SignaturOk
.bodysignatur_prüfen.#BodyOk.body_entschlüsseln.#entschlüsselterText
```

oder aber

```
maildaten_lesen.headersignatur_prüfen.#Fehlerinformation
.bodysignatur_prüfen.#Bodyfehler.body_entschlüsseln.#entschlüsselterText
```

In jedem Fall läuft also dieselbe Prozedur des Prüfens und Entschlüsselns ab. Der Unterschied liegt in den Ausgabeereignissen. Die Ausgabeereignisse werden in Form von Zusatzinformationen in den E-Mails hinzugefügt.

In jedem Fall erhält der Empfänger also eine E-Mail. Diese kann im Fehlerfall dann jedoch entsprechende Meldungen enthalten und im schlimmsten Fall einen fehlerhaft entschlüsselten Text.

5.5.2 Ablaufmodell des SMG-Send

ABBILDUNG 24. Lifecyclemodell: SMG-Send

```
liefecycle SMG-Send: (Senden)

Senden      =     maildaten_lesen.signiere_header.verschlüssle_body.
                  signiere_body.versenden
```

Beim Senden können weniger Fehlerfälle auftreten. Ist das SMG-Send erste einmal richtig installiert und konfiguriert, benötigt es zum Versenden nur seinen eigenen privaten Schlüssel und den öffentlichen Schlüssel des SMS. Damit kann die E-Mail in jedem Fall gefiltert und versendet werden.

5.5.3 Ablaufmodell des SMS

ABBILDUNG 25. Lifecyclemodell: SMS

```
liefecycle SMS: (Weiterleiten)

Weiterleiten    =     maildaten_lesen.headersignatur_prüfen.(
                      #FehlerAnAbsender|bodysignatur_prüfen.(
                      #FehlerAnAbsender|body_entschlüsseln.
                      empfängeradressen_auflösen.(signiere_header.
                      verschlüssle_body.signiere_body.versenden)*
                      ))
```

Interessanter wird das Verhalten des SMS. Dieser hat etwas komplere Abläufe, die sich auch durch ein komplexeres Ablaufmodell darstellen.

Sollte die Prüfung der Headersignatur hier fehlschlagen, wird eine Fehlermeldung an den Absender zurückgesendet und das Programm beendet. Nur, wenn die Headersignatur intakt ist, wird die Bodysignatur geprüft. Ein Fehler hier führt wiederum zu einem Abbruch. Ist die E-Mail aus kryptographischer Sicht korrekt, werden die Empfängeradressen aufgelöst. Wie bereits erwähnt, wird für jeden Empfänger erneut der Header signiert, der Body verschlüsselt und signiert und die so entstandene E-Mail versendet. Dies ist im Lifecyclemodell dadurch gekennzeichnet, daß hinter der Klammerung ein Stern (“*”) angegeben ist. Dieser Stern bedeutet, daß die angegebene Sequenz Null oder beliebig oft Mal durchgeführt wird. Der Fall Null tritt genau dann auf, wenn die Empfängeradresse nicht existiert. Denkbar wäre, in diesem Fall auch eine E-Mail an den Absender zu verschicken.

Bei dieser Entscheidung stehen sich Benutzerfreundlichkeit und Sicherheit gegenüber. Keine Antwort zu geben erhöht insofern die Sicherheit, als das nicht ausprobiert werden kann, welche Adressen es gibt, um so evtl. Rückschlüsse auf die Organisationsstruktur zu machen. So könnte man alle denkbaren Stellenzeichen der Verwaltung durchprobieren und jeweils auf der Netzwerkleitung ausspionieren, wer sich tatsächlich hinter diesen Stellen verbirgt.

Es hängt von der Firmenpolitik ab, ob der eine oder andere Weg gegangen wird. Dementsprechend ändert sich das Ablaufmodell.

5.6 Operationsmodelle

Ich habe mir mit Hilfe der Objektmodelle einen Überblick über die zu entwerfenden Objekte verschafft, mir mit Hilfe der Timeline-Diagramme die Systemgrenzen verdeutlicht und mir Beispiele verschafft, mit denen ich Ablaufmodelle erstellen konnte. Ich habe mir also deutlich gemacht, welche Operationen in welcher Reihenfolge in meinem System auftauchen.

Es wird nun Zeit, möglichst präzise darzustellen, wie sich die Operationen verhalten. Dieses Verhalten von Operationen wird in der Fusion Methode mit Hilfe der Operationsmodelle modelliert. Jede Operation bekommt eine Art Karteikarte, auf der kurz und präzise stehen soll, wozu diese Operation dient und was als Eingaben und Ausgaben erwartet wird.

Die Felder im Einzelnen:

Im Feld Operation steht der Operationsname. Im Feld Beschreibung folgt eine informelle oder formale Beschreibung der Operation. Im Feld “Nur Lesen” sind die Variablen und Attribute aufgeführt, auf die aus der Operation heraus lesend zugegriffen wird. In “Ändern” sind solche Variablen enthalten, die von der Operation heraus auch geschrieben werden. Ganz besonders interessant ist auch das Feld “Senden”. Hier werden alle Klassen aufgeführt, an die die Operation Nachrichten sendet. Oder anders ausgedrückt: Hier stehen alle Methoden von Klassen, die von dieser Operation aus aufgerufen werden. Dieses Feld ist besonders hilfreich bei der Implementierung und der Fehlersuche. Zunächst kann man die Operationen implementieren, die selbst keine anderen Operationen aufrufen. Bei der Fehlersuche verrät dieses Feld, in welchen aufgerufenen Operationen der Fehler evtl. zu finden sein könnte.

“Angenommen” beinhaltet die Vorbedingung für diese Operation. Jede Operation macht bestimmte Annahmen über den Zustand des Systems oder die Art der übergebenen Parameter o.ä. Wird die Operation aufgerufen, ohne daß diese Annahmen erfüllt sind, kann dies zu einem undefinierten Zustand oder einem Fehler führen. Ein Beispiel: Die Operation “formatiere” der Klasse Datenträger macht die Annahme, daß ein unbeschriebener Datenträger eingelegt ist. Wird gegen diese Annahme ein beschriebener Datenträger eingelegt, führt das Formatieren zum Datenverlust.

Das Feld “Ergebnis” beschreibt im Gegensatz zu “Angenommen” den Zustand des Systems nach der Ausführung. Es beinhaltet die Nachbedingung. Hier ist nicht nur beschrieben, was die Rückgabewerte beinhalten und was mit den Daten der Klasse geschehen ist sondern auch, welche Randeffekte diese Operation auslöst und wie sich der Zustand des Systems durch Ausführung der Operation verändert.

Ich möchte hier exemplarisch einige Operationen der Klassen Pkcrypt und Mail darstellen, da es sich hier um die wichtigsten Klassen dieses Projektes handelt. Es bietet sich an, bei Beschreibungen in den folgenden Kapiteln hier noch einmal nachzuschlagen, um das exakte Verhalten der beschriebenen Operationen noch einmal nachzulesen. Die hier nicht aufgeführten Operationen befinden sich zum großen Teil mit auf dem Datenträger zum Programmtext.

5.6.1 Operationen der Klasse Pkcrypt

Es ist schwierig für Pkcrypt, korrekte Operationsgraphen zu erstellen, da sich hinter Pkcrypt verschiedene kryptographische Methoden verstecken können. Zum Teil benötigen diese für den Zugriff auf die privaten oder öffentlichen Schlüssel verschiedene Attribute. Diese sind in den Operationsgraphen nicht angegeben. Dementsprechend sind Vorbedingungen, wie: “Der private Schlüssel existiert und ist mittels der gegebenen PIN zugreifbar.” zu verstehen. Im Falle einer PEM-Verschlüsselung mit Schlüsselverwaltung auf einem Massenspeicher (ohne Smartcard) würde dies bedeuten, daß die Attribute me, pin und pkpath gelesen würden und korrekte Daten enthalten müßten. Klassen, Relationen, Operationen sowie Attribute usw. sind im Datenlexikon erläutert. Das Datenlexikon wird, so sieht es die Fusion Methode vor, über die Dauer des gesamten Entwicklungsprozesses erstellt und gewartet. Das

von mir geführte Datenlexikon befindet sich im Anhang. Wird keine interne PEM-Verschlüsselung vorgenommen, sondern das externe PEM-Programm ausgeführt, müßte auch pempath korrekt gesetzt sein. Bei PGP stellt sich das wieder anders dar.

ABBILDUNG 26. Operationsgraph: Pkcrypt.signieren

Operation:	signieren
Beschreibung:	Signiert den gegebenen Text mit dem eigenen privatem Schlüssel
Nur Lesen:	SUPPLIED in : String
Ändern:	SUPPLIED out : String
Senden:	-
Angenommen:	Der private Schlüssel existiert und ist mittels der gegebenen PIN zugreifbar.
Ergebnis:	out enthält den signierten Text in

ABBILDUNG 27. Operationsgraph: Pkcrypt.pruefen

Operation:	pruefen
Beschreibung:	Überprüft den Text auf Integrität.
Nur Lesen:	SUPPLIED in : String
Ändern:	SUPPLIED out : String
Senden:	-
Angenommen:	Der öffentliche Schlüssel des Absenders ist bekannt.
Ergebnis:	WENN Integrität gewährt: Operation liefert TRUE, out ist der Originaltext. SONST: Operation liefert FALSE.

ABBILDUNG 28. Operationsgraph: Pkcrypt.verschlüsseln

Operation:	verschlüsseln
Beschreibung:	Verschlüsselt einen gegebenen Text für einen gegebenen Empfänger.
Nur Lesen:	SUPPLIED to : String, SUPPLIED in : String
Ändern:	SUPPLIED out : String
Senden:	-
Angenommen:	Der öffentliche Schlüssel des Empfängers to existiert.
Ergebnis:	out enthält den für to verschlüsselten Text.

ABBILDUNG 29. Operationsgraph: Pkcrypt.erzeuge_x-signatur

Operation:	erzeuge_x-signatur
Beschreibung:	Erzeugt aus den Signaturdaten eines gegebenen Kryptoformats einen Headerzeile-Body, der RFC 822 konform ist.
Nur Lesen:	SUPPLIED in : String
Ändern:	SUPPLIED out : String
Senden:	-
Angenommen:	in ist im erwartetem Format der jeweils durch den pktype identifizierten Kryptomethode.
Ergebnis:	out enthält den umgewandelten String.

ABBILDUNG 30. Operationsgraph: Pkcrypt.extrahiere_x-signatur

Operation:	extrahiere_x-signatur
Beschreibung:	Erzeugt aus dem Datenformat, das durch erzeuge_x-signatur() generiert wurde wieder eine Standardsignatur.
Nur Lesen:	SUPPLIED in : String
Ändern:	SUPPLIED out : String
Senden:	-
Angenommen:	in ist im erwartetem Datenformat.
Ergebnis:	out enthält eine Signatur im für diese Kryptomethode üblichen Format.

5.6.2 Operationen der Klasse Mail

ABBILDUNG 31. Operationsgraph: Mail.maildaten_lesen

Operation:	maildaten_lesen
Beschreibung:	Liest als Eingabestrom eine RFC konforme E-Mail samt Header und Body ein.
Nur Lesen:	-
Ändern:	h : Header, b : Body
Senden:	Header:{header_lesen}, Body:{body_lesen}
Angenommen:	Eingabestrom liefert exakt eine Mail mit Header und Body und ist EOF terminiert.
Ergebnis:	Header und Body sind in den Variablen h und b eingelesen.

ABBILDUNG 32. Operationsgraph: Mail.versenden

Operation:	versenden
Beschreibung:	Versendet E-Mail mittels gegebenen externem Programm.
Nur Lesen:	h : Header, b : Body, mailcmd : String
Ändern:	-
Senden:	-
Angenommen:	h und b enthalten gültigen Header und Body. Es ist ein externes Mailprogramm definiert.
Ergebnis:	E-Mail wird versendet.

ABBILDUNG 33. Operationsgraph: Mail.signiere_header

Operation:	signiere_header
Beschreibung:	Erzeugt eine normalisierte Form des Headers und läßt diesen signieren. Setzt dann die X-Signature Zeile.
Nur Lesen:	-
Ändern:	h : Header;
Senden:	Pkcrypt:{signieren}
Angenommen:	Der Header h enthält gültige Headerzeilen. Auf den privaten Schlüssel kann zugegriffen werden.
Ergebnis:	Der Header ist um die Zeile X-Signature erweitert oder X-Signature wurde ersetzt.

ABBILDUNG 34. Operationsgraph: Mail.headersignatur_pruefen

Operation:	headersignatur_pruefen
Beschreibung:	Prüft, ob die Signatur in X-Signature dem normalisierten Header entspricht.
Nur Lesen:	h : Header;
Ändern:	-
Senden:	Pkcrypt:{pruefen}
Angenommen:	h enthält einen gültigen Header inkl. X-Signaturezeile.
Ergebnis:	WENN Integrität des Headers gewährt DANN Operation liefert TRUE SONST Operation liefert FALSE.

ABBILDUNG 35. Operationsgraph: Mail.verschlüssle_body

Operation:	verschlüssle_body
Beschreibung:	Ersetzt den Mailbody durch den für einen gegebenen Empfänger verschlüsselten Mailbody.
Nur Lesen:	SUPPLIED empfänger: String
Ändern:	b : Body;
Senden:	Pkcrypt:{verschlüsseln}, Body:{löschen, hinzufügen}
Angenommen:	Es befinden sich Textzeilen im Mailbody.
Ergebnis:	Body enthält verschlüsselten Text verschlüsselt mit öffentlichem Schlüssel des Empfängers.

ABBILDUNG 36. Operationsgraph: Mail.signiere_body

Operation:	signiere_body
Beschreibung:	Signiert den Mailbody mit dem eigenen privatem Schlüssel.
Nur Lesen:	-
Ändern:	b : Body;
Senden:	Pkcrypt:{signieren}, Pkcrypt:{löschen, hinzufügen}
Angenommen:	Der private Schlüssel ist zugänglich. b enthält Text, der signiert werden kann.
Ergebnis:	b enthält den signierten Text.

ABBILDUNG 37. Operationsgraph: Mail.bodysignatur_prüfen

Operation:	bodysignatur_prüfen
Beschreibung:	Überprüft, ob die Signatur im Mailbody gültig ist und die Integrität des Body nicht verletzt wurde.
Nur Lesen:	-
Ändern:	b : Body;
Senden:	Pkcrypt:{prüfen}, Body:{löschen, hinzufügen}
Angenommen:	b enthält einen signierten Body. Der öffentliche Schlüssel des Absenders kann gelesen werden.
Ergebnis:	b enthält unsignierten Body. WENN Integrität ok DANN Operation liefert TRUE SONST Operation liefert FALSE.

ABBILDUNG 38. Operationsgraph: Mail.body_entschlüsseln

Operation:	body_entschlüsseln
Beschreibung:	Ersetzt den Body b durch den entschlüsselten Text von b.
Nur Lesen:	SUPPLIED InfoText : String;
Ändern:	b : Body;
Senden:	Pkcrypt:{entschlüsseln}, Body:{löschen, hinzufügen}
Angenommen:	Privater Schlüssel ist zugänglich. b enthält einen für den Empfänger verschlüsselten Text.
Ergebnis:	b ist ersetzt durch den InfoText gefolgt vom entschlüsselten Body.

5.6.3 Operationen weiterer Klassen

Die Operationsgraphen weiterer Klassen sowie die hier nicht aufgeführten Operationsgraphen der Klassen Pkcrypt und Mail können der Sourcecode-Dokumentation meiner Arbeit entnommen werden.

Die hier gezeigten Operationsgraphen zeigen die meiner Meinung nach wichtigsten Operationen und veranschaulichen das Prinzip der Entwicklungsarbeit an dieser Stelle.

KAPITEL 6

Wiederverwendbare Module (SME Bibliothek)

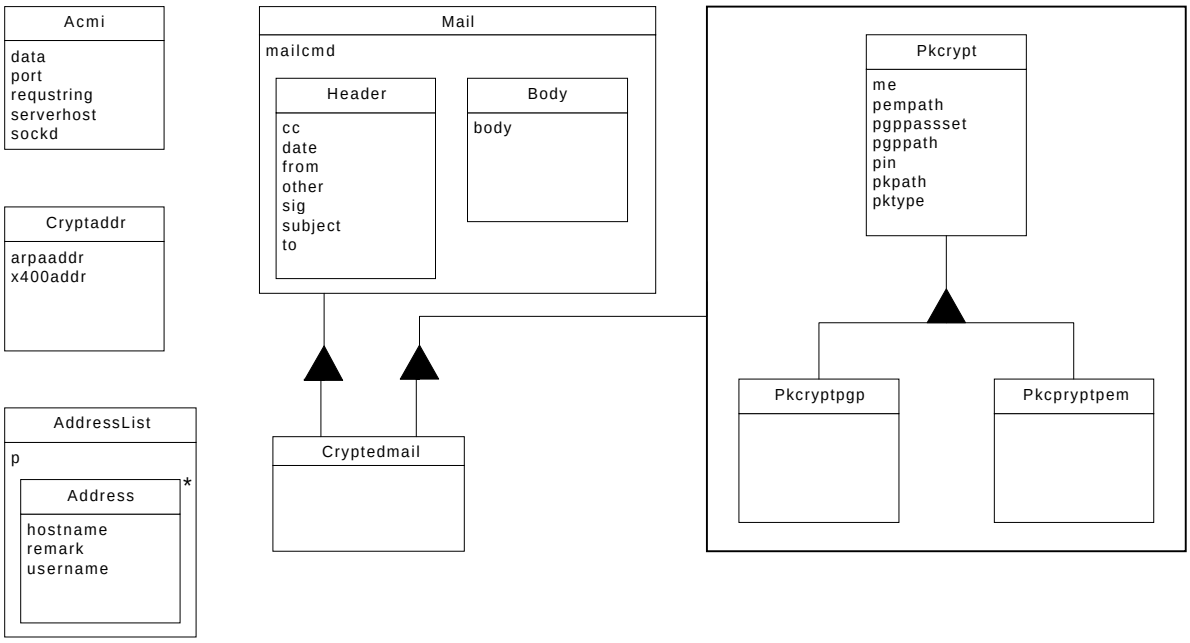
Wie sich in der Analysephase gezeigt hat, existieren diverse Klassen und Operationen, die für alle SME-Komponenten gleich sind. Das wirft die Frage auf, inwiefern diese Komponenten evtl. auch für andere Projekte wiederverwendbar sind. Wiederverwendbarkeit ist eines der großen Schlagworte in der objektorientierten Entwicklung. Am Beispiel der SME-Bibliothek werde ich versuchen darzustellen, wie eine Klasse beschaffen sein muß, um wiederverwendbar zu sein. Daß Klassen, die komplexere Datentypen, wie Listen, Stacks oder assoziative Arrays beinhalten (siehe STL Bibliothek [Amm97]), wiederverwendbar sind, steht außer Frage. Hierbei handelt es sich um Datentypen, deren Funktion in der Informatik allgemein bekannt sind und die semantisch mit verschiedenen Problemen aus der Informatik in Verbindung gebracht werden können. Wie sieht es nun aber mit Klassen wie “Pkcrypt”, “Acmi” oder “Mail” aus?

“Pkcrypt” ist die Repräsentation einer Public-Key Kryptographie mit einfachen Operationen, wie “verschlüsseln”, “signieren”, “prüfen” usw. Das klingt auf jeden Fall nach einem wiederverwendbaren Modul.

“Acmi” ist die Repräsentation eines Zugriffs auf das Access Control Modell. Eine solche Klasse kann nur in einem sehr begrenzten Kontext benutzt werden. Aber mit Sicherheit ist diese Klasse für jeden Entwickler interessant, der eine Anbindung seiner Software an das ACMI sucht. Es wäre also durchaus sinnvoll, diese Klasse dem ACM-Software-Paket hinzuzufügen, um eine Anbindung leichter zu machen. Dazu müßte die Klasse jedoch so weit erweitert werden, daß sie genauso allgemein ist, wie das Protokoll, das das ACMI zur Verfügung stellt. Zur Zeit sind die vom SME genutzten Funktionen implementiert. Ein allgemeines ACMI-Objekt macht nur dann als Bestandteil des ACM-Paktes Sinn, wenn alle Funktionen des ACMI darüber auch nutzbar sind.

Betrachten wir nun die Klasse “Mail”. Das Ergebnis einer genaueren Betrachtung lautet “teils, teils”. Eine Mail nach RFC 822 einlesen (also parsieren) oder verschicken zu können, den Body austauschen zu können etc. klingt wie eine Aufgabe, die sicherlich öfter benötigt werden könnte und zwar für jedes Programm, das irgend etwas mit E-Mails zu tun hat. Aber was ist mit Operationen, wie “erstelle_x-signatur”? Diese Funktionen lassen sich nur im Umfeld des SME benutzen. Auch die Verschlüsselungsfunktionen sind für eine einfache Mailklasse etwas speziell. Die von mir analysierte Klasse Mail ist in der Realität also ein Art spezialisierter Mail. Und “Spezialisierung” und “Generalisierung” sind doch bekannte Vorgänge in der objektorientierten Modellierung. Ich trenne nun die Aufgaben einer Standardmail von dieser speziellen Mail und entwerfe die Klasse “Cryptedmail”, die alle Eigenschaften einer gewöhnlichen Mail besitzt, jedoch zusätzlich kryptographische Funktionen nutzen kann.

ABBILDUNG 39. Vererbungsgraph: SME Bibliothek



Die Abbildung zeigt die Sammlung von Objektklassen für die Bibliothek. Die Klassen “Acmi” und “Cryptaddr” stehen in keiner weiteren Beziehung zu irgendeiner Klasse in der Bibliothek. “Address-List” ist eine Liste für eine Menge von “Address”-Objekten. “Mail” enthält einen “Header” und einen “Body”. “Pkcrypt” ist beim heutigen Stand der Entwicklung entweder ein “Pkcryptpgp” oder ein “Pkcryptpem”. “Cryptedmail” erbt sowohl von “Mail” als auch von “Pkcrypt”, stellt jedoch nur die Methoden von “Mail” zur Verfügung. Sinnvoller wäre hier eine andere Beziehung (z.B. friend) zu “Pkcrypt”. Das ist jedoch vor allem ein Problem der Implementierung.

Die SME Bibliothek als Ganzes ist nur für die Verwendung in den SME Programmen gedacht. SMS, SMG und SPT greifen auf Klassen dieser Bibliothek zu. Einige Klassen dieser Bibliothek sind jedoch auch für die Wiederverwendung in anderen Projekten geeignet.

Um eine bessere Wiederverwendbarkeit zu erlangen, änderte ich auch den Entwurf der AddressList etwas, die nun eine einfache Adresse, bestehend aus dem Benutzernamen, dem Rechnernamen mit Domain und der Bemerkung in einem Adressfeld besteht. Die Klasse Cryptaddr existiert für sich allein und dient nur noch dem Zerlegen der Rückgaben aus dem ACMI. Aus der Implementierung ergab sich, daß eine Liste dieser Cryptaddr-Objekte nicht unbedingt nötig ist.

Weiterführende Informationen zu den einzelnen Attributen der Objekte befinden sich im Datenlexikon.

6.1 Objektinteraktionsgraphen

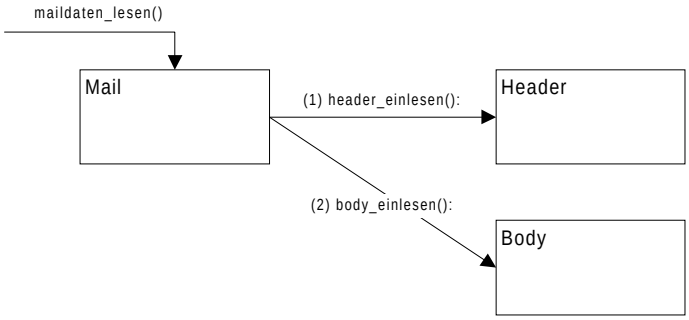
Nachdem die Analyse des Systems nach der Fusion Methode nun abgeschlossen ist, komme ich zur Entwurfsphase. Wie bereits erwähnt, ist diese strikte Trennung vor allem für die Gliederung meiner Arbeit interessant. Während meiner Arbeit gingen die Phasen fließend ineinander über. So stellt man zum Teil erst beim Entwurf fest, was offensichtlich bei der Analyse vergessen wurde und muß dann beide Modelle aneinander angleichen oder bestimmte Schritte einfach wiederholen.

Der Objektinteraktionsgraph stellt, wie der Name schon sagt, die Kommunikation der Objekte untereinander dar. Im Gegensatz zum Timeline Diagramm ist die Darstellung hier jedoch kein Beispiel,

sondern stellt alle möglichen Kommunikationswege dar. Es ist sozusagen ein Zusammenfassen des gewonnenen Wissens aus dem Objektmodell, dem Ablaufmodell und den Operationsgraphen und ein weiter wichtiger Schritt hin zum Design der Klassen.

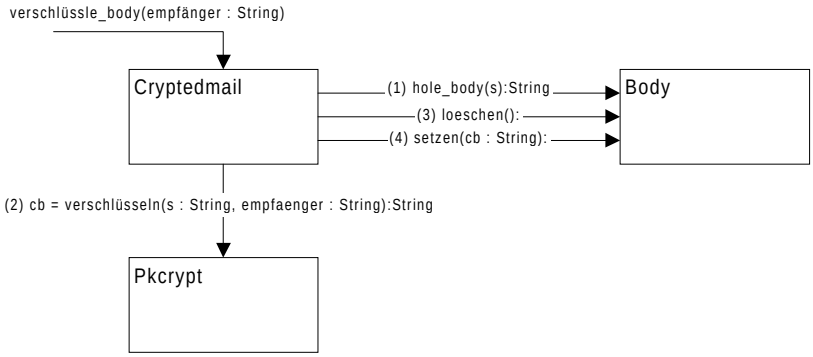
Im folgenden einige Beispiele aus den Objektinteraktionsgraphen meines Entwurfs:

ABBILDUNG 40. Objektinteraktionsgraph: Mail.maildaten_lesen()



Ruft das System die Methode Mail.maildaten_lesen() auf, so richtet diese Funktion jeweils eine Anfrage header_einlesen() und body_einlesen() an die in Mail enthaltenen Objekte Header und Body. Wie aus dem Objektmodell bekannt ist, sind Header und Body zwei Objekte, die selbst Attribute von Mail sind und damit auch von Cryptedmail. Die Reihenfolge der Nachrichten ist durch die in der Klammer vor den Methodennamen angegebenen Zahlen festgelegt. Es wird stets zuerst der Header gelesen und dann der Body, was der Lesereihenfolge des Eingabestroms entspricht. Die E-Mail ist so definiert, daß zunächst der Header übertragen wird und nach der ersten Leerzeile dann der Body folgt. Die Methoden von Mail stehen dank der Vererbung auch Cryptedmail zur Verfügung.

ABBILDUNG 41. Objektinteraktionsgraph: Cryptedmail.verschlüssle_body()

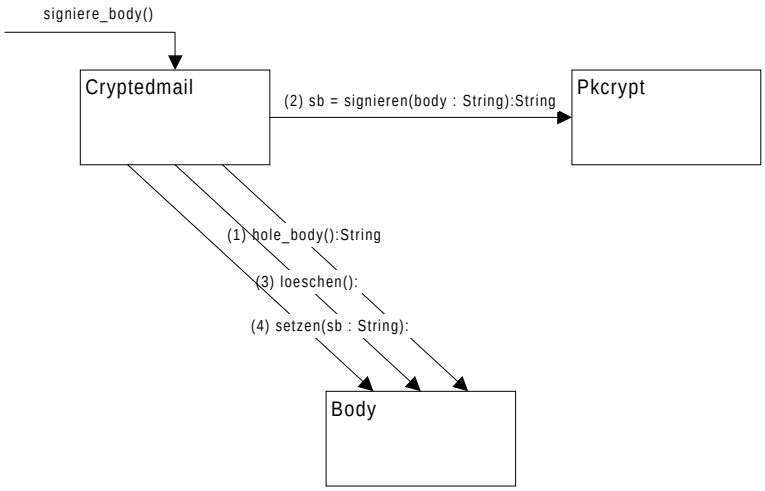


Wie man an diesem Beispiel sehen kann, bietet die Erstellung der Objektinteraktionsgraphen noch einmal die Möglichkeit, zu überdenken, von welchem Objekt aus die Funktionalität implementiert wird. Im abgebildeten Objektinteraktionsgraphen wird die Methode verschlüssle_body() der Klasse Mail aufgerufen. Diese ruft dann Pkcrypt.verschlüsseln() auf und übergibt dabei den Body, der verschlüsselt werden soll. Dann löscht Mail den Body und setzt ihn neu also ersetzt ihn durch den Verschlüsselten Body (das Ergebnis der Verschlüsselung von Pkcrypt.verschlüsseln()).

Genauso hätte man den Entwurf auch dahingehend ändern können, daß Mail.verschlüssle_body() eine Methode Body.verschlüsseln() aufruft und Body selbst die Verschlüsselung durch Pkcrypt.verschlüsseln() durchführen läßt.

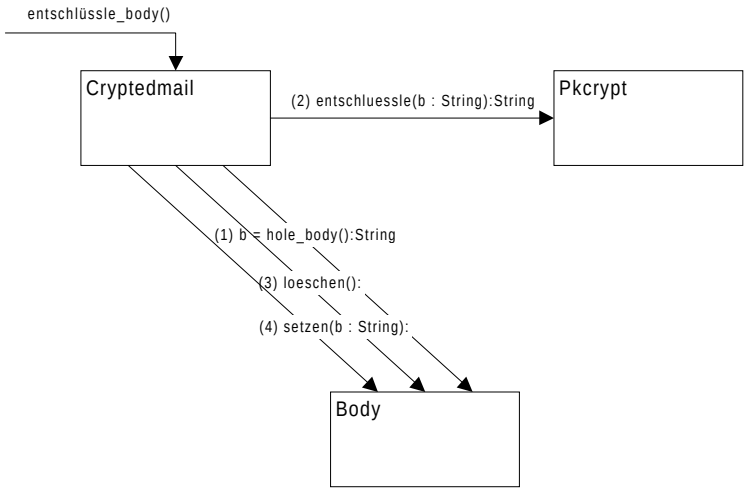
Meine Designvorstellung geht jedoch in die Richtung, daß Header und Body nichts mit Verschlüsselungen zu tun haben sollen. Sie sollen einfach, wie die Klasse Mail eine allgemeine Mail repräsentieren und ich möchte nicht noch Konstrukte, wie Cryptedbody und Cryptedheader einführen.

ABBILDUNG 42. Objektinteraktionsgraph: Cryptedmail.signiere_body()



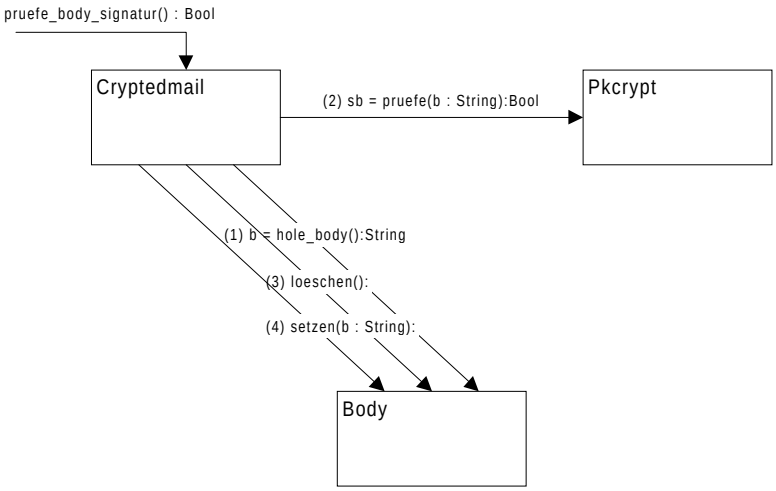
Die Methode Mail.signiere_body() funktioniert sehr ähnlich wie verschlüssle_body(), nur daß hier an Pkcrypt außer dem Body-String keine weiteren Parameter (wie Empfänger) übergeben werden. Zum Signieren reicht der eigene private Schlüssel. Dieser ist als Attribut in Pkcrypt enthalten und wird beim Erstellen eines neuen Pkcrypt Objektes als Parameter mit angegeben.

ABBILDUNG 43. Objektinteraktionsgraph: Cryptedmail.entschlüssle_body()



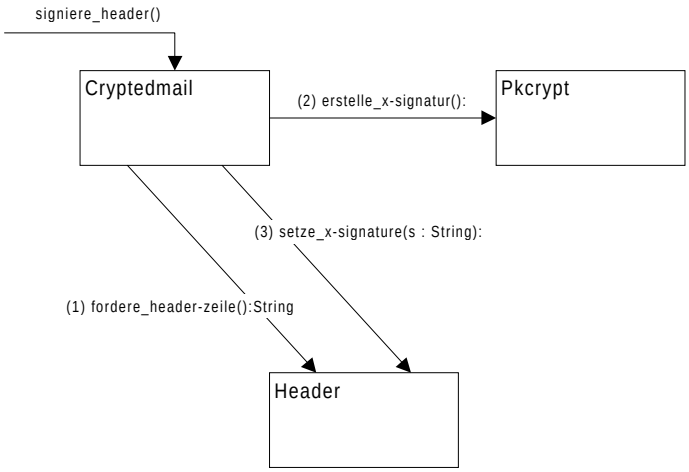
Das Entschlüsseln läuft analog zum Verschlüsseln des Body.

ABBILDUNG 44. Objektinteraktionsgraph: Cryptedmail.prüfe_body_signatur()



Der Rückgabewert von Pkcrypt.pruefe() wird als Rückgabewert von prüfe_body_signatur() an den Sender dieser Nachricht zurückgeliefert.

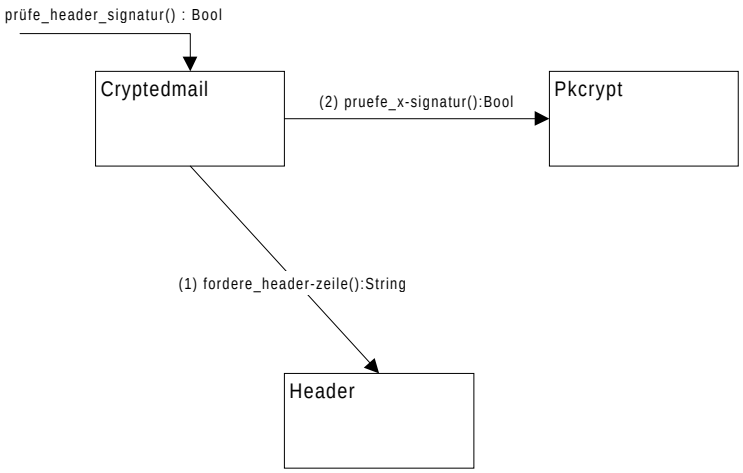
ABBILDUNG 45. Objektinteraktionsgraph: Cryptedmail.signiere_header()



Den Objektinteraktionsgraphen von signiere_header() habe ich, wie auch die Darstellungen im Timelinediagramm stark vereinfacht. In Wirklichkeit werden hier die Zeilen “To:”, “From:”, “Subject:” und “Date:” von der Klasse Header angefordert. Es wäre mir wiederum zu speziell gewesen, das fertige 4-Tupel aus einer Funktion zu ziehen, da dies wiederum nur im SME, speziell in der Klasse Cryptedmail von Bedeutung ist.

Die Methode fordere_header-zeilen() existiert also in Realität nicht. Sie ist implementiert als eine Sequenz von fordere_to(), fordere_from() usw.

ABBILDUNG 46. **Objektinteraktionsgraph: Cryptedmail.prüfe_header_signatur()**



Auch hier versteckt sich hinter fordere_header-zeile() die Sequenz von Methodenaufrufen. Zu beachten ist, daß hier der Vergleichstext für die X-Signatur neu zusammengestellt wird, bevor er geprüft wird. Aus diesem Grund ist es wichtig, daß der Vergleichsheader (der nur zur Überprüfung erstellt wird) nach dem gleichen Algorithmus zusammengestellt wird, wie beim Signieren des Headers, da bei jeder Zeichenabweichung die Prüfung fehlschlagen würde.

6.2 Die Klasse “Config”

Bei der gesamten Analyse und Planung des Projektes habe ich ein Problem immer weiter aufgeschoben; die Frage nach der Konfiguration der entsprechenden Tools. Aber welcher Prototyp findet überhaupt Anwendung, wenn an jeder Stelle ein neuer Compilerlauf nötig ist, um die Installation an eine bestimmte Maschine anpassen zu können. Vor allem aber auch bei unseren Tests machte es sich unangenehm bemerkbar, daß allein das Ändern des Schlüssel-Identifizierers eine Neuübersetzung forderte.

Aus diesem Grund führe ich zusätzlich die Klasse “Config” ein, die eine Konfiguration repräsentieren soll. Sie zeichnet sich durch die folgenden Operationen aus, die ich an dieser Stelle noch einmal als Operationsmodell darstellen möchte:

6.2.1 Beschreibung der Klasse “Config”

Die Klasse “Config” soll Dateien lesen, die jeweils Zeilen der Form:

`<variable>=<wert>`

enthalten. Dabei soll <variable> ein Bezeichner sein, der aus einer beliebigen Zeichenkette besteht, die weder Leerzeichen, Tabulatoren noch “=” Zeichen enthält. <wert> kann jede beliebige Zeichenkette sein, die kein Zeilenende beinhaltet.

Damit können alle einzeiligen Werte zur Konfiguration herangezogen werden. Ist es nötig, Zeilenwechsel zu konfigurieren, kann in dem Programm, das “Config” benutzt, intern eine Funktion, wie “sprintf()” benutzt werden, die Zeilenenden durch Umschreibungen erzeugen kann.

Ferner ist es möglich, mit “Config” auch Umgebungsvariablen einzulesen, was vor allem zum Testen und für bestimmte Shellskripte interessant sein kann.

Die Abfrage kann über verschiedene Methoden realisiert werden, die bereits eine Typenwandlung vornehmen. Welcher Typ jeweils zu erwarten ist, hängt von der Anwendung ab. Das Programm selbst muß untersuchen, ob es sich beim Rückgabewert um einen gültigen Wert handelt bzw. ob ein Typenkonflikt vorliegt. Abfragen von Werten mit falschen Typen führen zu undefinierten Rückgabewerten.

Die Implementierung soll auf Basis einer STL Map stattfinden. STL steht für “Standard Template Library” und ist eine Sammlung von Klassen, die für die meisten C++-Compiler zur Verfügung steht. “Map” ist eine Klasse aus der STL Bibliothek dessen Idee die eines assioziativen Arrays ist, also ein Feld, dessen Argumente nicht über einen Zahlenindex angesprochen werden können, sondern über eine Zeichenkette. Auf einer solchen Map sind Operationen, wie z.B.

```
m["Erster"]=5;
m["Zweiter]=10;
```

möglich. Welchen Typ der Inhalt der Map hat und welchen Typ die Identifizierer der Map haben, sind dem Anwender der STL-Bibliothek überlassen [Amm97].

In meinem Fall benutze ich als Identifizierer Zeiger auf Zeichenfelder (char*) und als Inhalt Zeichenketten (String).

In der Praxis sieht eine Konfigurationsdatei z.B. so aus:

ABBILDUNG 47. Beispieldatei: winspt.ini

```
myname=C=de,O=TUB,CN=Thomas.Hildmann
mypin=12345
smtp_port=25
pop3_port=110
smtp_server=lexx.mash-gmbh.com
pop3_server=lexx.mash-gmbh.com
```

Diese Datei baut eine Map auf, die das folgende Aussehen besitzt:

ABBILDUNG 48. Konfiguration als Map

```
m["myname"]="C=de,O=TUB,CN=Thomas.Hildmann"
m["mypin"]="12345"
m["smtp_port"]="25"
m["pop3_port"]="110"
m["smtp_server"]="lexx.mash-gmbh.com"
m["pop3_server"]="lexx.mash-gmbh.com"
```

Mit Hilfe des Methodenaufrufes `name = Config.getChar(“myname”)` läßt sich der Wert `"C=de,O=TUB,CN=Thomas.Hildmann"` aus der oben abgebildeten Konfiguration auslesen.

Die Variable “mypin” ist in dieser Konfigurationdatei noch enthalten, weil im aktuellen Prototypen des Windowsprogramms noch keine PIN-Abfrage implementiert ist. Aus diesem Grund wird zunächst die PIN aus der Konfigurationsdatei gelesen.

6.2.2 Operationsmodell

Im nächsten Schritt werde ich die Operationsgraphen der Klasse “Config” vorstellen, um einen kleinen Überblick darüber zu geben, welche Methoden diese Klasse zur Verfügung stellt und was diese bewirken.

Ich bilde hier exemplarisch den typischen Abschnitt einer Software-Entwicklung innerhalb einer Software-Entwicklung ab. Die Klasse “Config” ist für sich hier analysiert und entworfen worden. Sie kann dann unabhängig getestet werden und im übergeordneten Entwicklungsprozeß als gegebenes Werkzeug angesehen werden.

ABBILDUNG 49. Operationsgraph: Config.löschen

Operation:	löschen
Beschreibung:	Setzt die Konfiguration auf die leere Konfiguration.
Nur Lesen:	-
Ändern:	cm : ConfigMap
Senden:	-
Angenommen:	-
Ergebnis:	cm ist leer.

ABBILDUNG 50. Operationsgraph: Config.lesen

Operation:	lesen
Beschreibung:	Liest die Konfiguration aus einer gegebenen Datei und füllt mit den Daten die ConfigMap cm.
Nur Lesen:	SUPPLIED filename : String
Ändern:	cm : ConfigMap
Senden:	-
Angenommen:	Datei, die durch filename beschrieben ist, enthält Konfigurationseinträge im korrekten Format.
Ergebnis:	cm enthält eine Abbildung der Konfiguration aus der Datei.

ABBILDUNG 51. Operationsgraph: Config.hinzufügen

Operation:	hinzufügen
Beschreibung:	Fügt die Einträge aus der Datei filename der ConfigMap cm hinzu.
Nur Lesen:	SUPPLIED filename : String
Ändern:	cm : ConfigMap
Senden:	-
Angenommen:	Datei, die durch filename beschrieben ist, enthält Konfigurationseinträge im korrekten Format.
Ergebnis:	cm enthält zusätzlich eine Abbildung der Konfiguration aus der Datei. Doppelte Einträge sind überschrieben.

ABBILDUNG 52. Operationsgraph: Config.existiert

Operation:	existiert
Beschreibung:	Liefert TRUE, falls ein Konfigurationseintrag mit der Bezeichnung desc existiert.
Nur Lesen:	SUPPLIED desc : String, cm : ConfigMap
Ändern:	-
Senden:	-
Angenommen:	-
Ergebnis:	WENN desc in cm enthalten DANN Operation liefert TRUE SONST Operation leifert FALSE.

Hinzu kommen diverse Operationen zum Abfragen der Inhalte bestimmter Variablen. Diese gehen jeweils davon aus, daß die Typen verträglich sind und liefern ansonsten undefinierte Rückgabewerte.

6.3 Klassenbeschreibungen

Als letzter Entwurfsschritt der Fusion Methode folgt nun die Klassenbeschreibung. In diese Notation gehen die Informationen aller vorhergegangener Diagramme ein. Von dieser Darstellung aus ist es nur noch ein sehr kleiner Schritt zu den Deklarationen in den Programmtexten.

Wie bereits in der Einleitung erwähnt, handelte es sich beim Projekt E2S um ein EU-Projekt. An diesem Projekt arbeiteten also Projektpartner aus verschiedenen europäischen Staaten. Aus diesem Grund stand fest, daß die Implementierung der Software zumindest im Sourcecode vollständig englisch sein sollte. Während ich bislang die Entwürfe weitgehend in deutsch gehalten habe, werde ich nun auf die im Sourcecode verwendeten englischen Begriffe zurückgreifen und diese auf deutsch dokumentieren.

6.3.1 Klasse Pkcrypt

Hier der relevante Ausschnitt aus dem Klassenentwurf für die Klasse Pkcrypt:

```
class Pkcrypt
  attribute variable me : exclusive String
  attribute variable pin : exclusive String
  attribute variable pkpath : exclusive String
  attribute variable pgppath : exclusive String
  attribute variable pempath : exclusive String
  attribute variable pgppath : exclusive String
  attribute variable pgppassset : exclusive bool
  attribute variable pktype : exclusive int

  method print()
  method sign(constant in : String, out : String)
  method mkxsig(constant in : String, out : String)
  method unmkxsig(constant in : String, out : String)
  method crypt(constant to : String, constant in : String, out : String)
  method decrypt(constant in : String, out : String)
  method check(constant in : String, out : String) : bool
  method setPin(constant pin : String);
  method getPin() : String
  method readPin()
endclass
```

Kurzbeschreibungen über die Repräsentation der Variablen und der Funktion der Methoden befinden sich im Anhang.

6.3.2 Klasse Mail

```
class Mail
  attribute variable mailcmd : exclusive String
  attribute variable h : exclusive Header
  attribute variable b : exclusive Body

  method getHeader() : Header
  method getBody() : Body
  method putHeader(constant mh : Header)
  method putBody(constant mb : Body)
  method forwardMail(constant to : String)
  method returnToSender()
  method setMailcmd(constant mailcmd : String)
  method read();
  method write();
endclass
```

In der eigentlichen Implementierung sind die Methoden read() und write() die Operatoren operator<< und operator>>.

6.3.3 Klasse Cryptedmail

```
class Cryptedmail isa Mail, Pkcrypt
  method checkHeaderSignature() : bool
  method checkBodySignature() : bool
  method decryptBody(constant infoText : String)
  method signHeader()
  method cryptBody(constant recipient : String)
  method signBody()
endclass
```

Das Schlüsselwort “isa” (ist ein) drückt die Vererbung aus. Das heißt, daß Cryptedmail sowohl ein Kind der Klasse Mail als auch ein Kind der Klasse Pkcrypt ist, so wie das im Vererbungsgraphen dargestellt war.

6.3.4 Klasse Adressliste

```
class AddressList
  attribute variable addresses : exclusive olist address

  method add(constant a : Address)
  method isempty() : bool
  method size() : long int
```

```
  method first() : Address
  method next() : Address
  method StringToAddressList(const s : String)
  method print()
  method read()
  method write
endclass
```

6.3.5 Klasse Adresse

```
class Address
  attribute variable username : exclusive String
  attribute variable hostname : exclusive String
  attribute variable remark : exclusive String

  method setAddress(constant s : String)
  method getUsername() : String
  method getHostame() : String
  method getRemark() : String
  method setRemark(constant rem : String)
  method isempty() : bool
  method read()
  method write()
endclass
```

6.3.6 Klasse Acmi

```
class Acmi
  attribute variable serverhost : exclusive String
  attribute variable requstring : exclusive String
  attribute variable data : exclusive String
  attribute variable port : exclusive int
  attribute variable sockd : exclusive int

  method request(protocol : String, user : String,
     object : String, method : String, target : String) : String
  method resolveAddr(user : String, target : String) : String
endclass
```

6.3.7 Klasse Body

```
class Body
  attribute variable body : exclusive String

  method clear()
  method addline(constant s : String)
  method insertline(constant s : String)
  method toString() : String
  method read()
  method write()
endclass
```

6.3.8 Klasse Header

```
class Header
  attribute variable to : exclusive String
  attribute variable from : exclusive String
  attribute variable cc : exclusive String
  attribute variable subject : exclusive String
  attribute variable date : exclusive String
  attribute variable sig : exclusive String
  attribute variable other : exclusive String

  method getXxx() : String
  method setXxx(Yyy : String)
  method setNow()
  method read()
  method write()
endclass
```

Dabei stehen die Methoden getXxx() und setXxx() für Get- und Set-Funktionen für jedes der Attribute.

6.3.9 Klasse Cryptaddr

```
class Cryptaddr
  attribute variable arpaaddr : exclusive String
  attribute variable x400addr : exclusive String

  method getArpaaddr() : String
  method getX400addr() : String
endclass
```

Wie sich bei der Implementierung herausstellen wird, werden die Methoden an einigen Stellen noch erweitert. So benötige ich für diverse Operationen in den Programmen z.B. den C++-Operator operator= für Zuweisungen oder es müssen Operatoren operator< bzw. operator> eingeführt werden. Für Details hierzu lohnt ein Blick in die Quelltexte. Für den Programmentwurf oder die Prinzipien, die hinter dem gesamten System stecken, sind diese Details jedoch eher Faktoren, die verwirren als Fakten, die genannt werden sollten.

KAPITEL 7

Entwurf der einzelnen Softwarekomponenten

Die Komponenten SMG und SMS sind durch den Entwurf der SME Bibliothek ausreichend beschrieben. Sie beinhalten keine Oberfläche oder sonstige Besonderheiten sondern rufen nur die beschriebenen Methoden auf. Die Konfiguration dieser Programme beschreibe ich in „Installation und Konfiguration“ auf Seite 107.

7.1 Secure Mail Proxy Tools (SPT)

Beim Entwurf der SPT müssen einige zusätzliche Anforderungen bedacht werden:

- Es soll nur ein Programm gestartet werden, da jedes Programm, das die Secude-Bibliothek benutzt, eine Meldung auf den Bildschirm bringt, die über die Urheberrechte der Bibliothek aufklärt. Es wird von den Benutzern als störend bzw. Fehler empfunden, mehrfach die gleiche Meldung bestätigen zu müssen.
- Trotzdem muß die Nebenläufigkeit erhalten bleiben, denn ggf. können gleichzeitig E-Mails empfangen und versendet werden.
- Es muß eine Benutzungsschnittstelle (GUI) entworfen werden, um die Aktionen der SPTs für den Benutzer transparent zu machen.

7.1.1 Multithreaded

Das Dilemma, ein Programm vs. Nebenläufigkeit löse ich durch Starten mehrerer Threads; das unter Windows übliche Vorgehen. Im Programmtext sieht das dann wie folgt aus:

```
void initializeThreads()
{
  DWORD dwThreadIdPop3;
  DWORD dwThreadIdSmtp;

  windows_startup();
```

```
  CreateThread(NULL, NULL, pop3Thread, NULL, 0, &dwThreadIdPop3);
  CreateThread(NULL, NULL, smtpThread, NULL, 0, &dwThreadIdSmtp);
}
```

Dabei sind pop3Thread() und smtpThread() zwei Funktionen, die als Einstiegspunkte dienen. Hinter jeder Funktion verbirgt sich die jeweilige Implementierung von POP3-Proxy bzw. SMTP-Proxy.

Es laufen letztendlich also drei Threads. Der initiale Thread, der weiter die Benutzungsoberfläche verwaltet, der POP3-Thread, der POP3 Anfragen weiterleitet und der SMTP-Thread, der den Proxy für die SMTP-Verbindung darstellt.

Die Kommunikation zwischen den jeweiligen Proxy-Threads und dem GUI-Thread findet über den gemeinsamen Datenbereich statt, den sich die Threads teilen. Das heißt, daß nach dem CreateThread() jedem der laufenden Threads weiter alle Variablen zur Verfügung stehen und damit auch die Zeiger auf die GUI-Objekte.

7.1.2 Die Oberfläche

Ziel der SPTs ist eigentlich, so wenig wie möglich aufzufallen. Das heißt, die Benutzerin bzw. der Benutzer sollen möglichst wenig von der Existenz des SPT mitbekommen. Trotzdem ist ein wichtiger Grundsatz der GUI-Gestaltung, daß die Benutzer über den Status des Programms aufgeklärt werden sollen. Es soll also mindestens erkennbar sein, daß das Programm gestartet ist. Verschwindet das WINSPT nach der Secude-Meldung in den Interna des Windows Betriebssystems ist das für die Benutzerinnen und Benutzer nicht besonders transparent. Deshalb erzeuge ich ein einfaches, unauffälliges Fenster, das allein die Existenz und die Bereitschaft des WINSPT anzeigt. Dieses Fenster kann ikonifiziert werden und stört den weiteren Arbeitsablauf nicht. Durch die Existenz des Fensters wird die Benutzerin / der Benutzer jedoch an die Funktionsbereitschaft des Programms erinnert.

ABBILDUNG 53. Hauptfenster des WINSPT

Das Fenster enthält in der Titelzeile die Symbole für das Maximieren und das Ikonifizieren. Die Schaltfläche zum Schließen des Fensters ist jedoch deaktiviert (graues statt schwarzes Kreuz). Damit kann das Programm auf herkömmliche Art nicht beendet werden.

Dies ist eine Vorsichtsmaßnahme, die verhindern soll, daß aus Versehen die Proxy-Tools beendet werden und damit der E-Mailverkehr unmöglich gemacht wird. Dies verhindert jedoch nicht ein gewolltes Beenden durch Drücken der Tastenkombination Strg-Alt-Entf und Beenden des Tasks:

ABBILDUNG 54. Fenster: Anwendung schließen

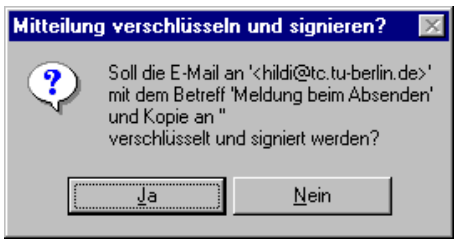
7.1.3 Meldungen bei Senden oder Empfangen von Mails

Das Signaturgesetz fordert, daß der Absender eine E-Mail darüber informiert wird, daß er gerade eine Unterschrift leistet. Genauso wird beim Empfang gefordert, daß die Existenz einer gültigen Unterschrift angezeigt wird [SigG].

Wie eine rechtliche Regelung bei der Verwendung eines SMGs aussehen könnte, habe ich im Rahmen meiner Diplomarbeit nicht untersucht. Evtl. ist es ausreichend, den Teilnehmern zur Kenntnis zu geben, daß alle ihre Nachrichten signiert werden, es sei denn, sie stellen der Adresse ein "_"-Zeichen voran. Vielleicht muß auch noch eine Java-Application programmiert werden, die die Kommunikation mit dem SMG aufrecht erhält. Ich würde an dieser Stelle auf die Java-Technologie zurückgreifen, da ich an Rechner hinter einer SMG keine Anforderungen an die Plattform stellen wollte. Um nicht für jede Betriebssystem-Plattform ein neues Infoprogramm erstellen zu müssen, würde sich an dieser Stelle anbieten, auf Java zurückzugreifen, da Java-Programme dank ihres Virtual Machine[1]-Konzepts auf allen Plattformen lauffähig sind, für die es eine solche Virtual Machine gibt.

Beim Senden erhält die Benutzerin / der Benutzer eine Meldung der Form:

ABBILDUNG 55. Fenster: Mitteilung verschlüsseln und signieren?



Ein ähnliches Fenster teilt den Empfang einer verschlüsselten oder unverschlüsselten E-Mail mit.

Da vom Benutzer nicht verlangt werden kann, daß er bei einer größeren Anzahl von empfangenen Mails noch weiß, welche verschlüsselt und welche unverschlüsselt an ihn gesendet wurde und was bei welcher E-Mail auf dem Bildschirm stand, werden als erste Zeile in den Body Statusmeldungen von den Proxy-Tools hinzugefügt, die später eine Einordnung ermöglichen.

1. Es handelt sich hierbei um einen Byte-Code-Interpreter, der den vorübersetzten Java-Bytecode auf der jeweiligen Plattform ausführen kann.

Solche Statusmeldungen lassen sich zwar fälschen, indem man selbst eine Meldung in die erste Zeile schreibt. Nur stellen die Proxy-Tools, wie auch das SMG immer eine Meldung voran, so daß die Empfängerin bzw. der Empfänger die zwei Meldungen bemerken würde.

Wichtig ist jedoch, die Benutzerinnen und Benutzer darauf aufmerksam zu machen, daß das Auffinden von zwei Meldungen in der Mail darauf schließen läßt, daß hier ein Betrugsversuch vorliegt.

KAPITEL 8

Schnittstellen zu anderen Softwarekomponenten

Die Schnittstelle zu anderen Softwarekomponenten ist eines der größten Herausforderungen in einem größeren Softwareprojekt. Kaum ein Programm steht heute für sich allein. Fast alle Programme nutzen Dienste von anderen Programmen oder stellen ihnen zumindest Daten o.ä. zur Verfügung.

Während die Datenformate und Typisierungen innerhalb des eigenen Systems mit Hilfe einer guten Spezifikation und nicht zuletzt mit Hilfe von Typenprüfungen des Compilers und/oder Laufzeitsystems in den Griff zu bekommen sind, hat die Kommunikation mit anderen Softwarekomponenten eine ganz andere Qualität. Hier ist man auf eine gute Dokumentation der Protokolle angewiesen. Nur, wenn eine Schnittstelle über Schnittstellen-Definitionssprachen, wie z.B. IDL (Interface Definition Language) definiert ist, können Compiler helfen, zumindest die Typisierung zu prüfen.

In diesem Kapitel möchte ich auf die diversen Schnittstellen eingehen, die das SME zu anderen Softwarekomponenten hat bzw. haben kann.

8.1 Schnittstelle zum Access Control Model

Im Kapitel „Einbindung von rollenbasierter Zugriffskontrolle“ auf Seite 31 habe ich beschrieben, wie die Informationen des ACM zur Funktionserweiterung für E-Mails genutzt werden können. Das ACM kann zur Abbildung von Organisationsstrukturen verwendet werden. Ziel der Modellierung einer Organisation ist die Vereinfachung der Zugriffskontrolle im EDV-Bereich. Besonders interessant ist dies im Bereich WWW. Viele Programme bieten bereits Schnittstellen zum WWW an oder können dahingehend leicht erweitert werden. Das am PRZ der TU Berlin entwickelte Konzept versucht zudem, eine möglichst einheitliche Oberfläche zu implementieren und damit nicht nur eine Erleichterung in der Administration zu schaffen, sondern auch eine für die Benutzer.

Das ACM wird aber um so interessanter, je mehr Softwarekomponenten in der Lage sind, mit ihm zu kommunizieren und die Modellierungsmöglichkeiten zu nutzen. Die Integration des ACM in das SME ist nur eine Möglichkeit.

Die gesamte Arbeit am ACM war so angelegt, daß zum einen ein möglichst flexibles Modell entwikkelt werden sollte, weiterhin ein Interpreter (ACMI) und als dritte Komponente ein Generator für dynamische HTML-Seiten für das WWW. Der ACMI ist über eine Internet Socketverbindung erreichbar. Die Kommunikation findet über ein menschenlesbares Protokoll statt. Das heißt, daß alle Nachrichten zum ACMI und vom ACMI im ASCII-Format kodiert sind und es sich um Befehle,

Aufzählungen oder Strukturierungselemente handelt, die für Menschen verständlich sind, sofern diese das Protokoll und dessen Semantik kennen.

8.1.1 Das Protokoll des ACMI

Grundsätzlich antwortet der ACMI auf eine Anfrage jeweils mit einem Antwortelement. Das Anfrageformat ist jeweils so gewählt, daß alle für die Bearbeitung im ACMI nötigen Informationen enthalten sind. Das heißt, daß auch, wenn die Verbindung zum ACMI offen gehalten werden kann, das ACMI nicht zustandsorientiert zum anderen Gesprächsteilnehmer arbeitet. Es findet also kein Anmeldevorgang mit Authentifizierung statt, worauf die Anfragen folgen, sondern mit jeder Anfrage findet eine neue Authentifizierung statt.

Im Grunde handelt es sich bei allen Anfragen an das ACMI um Methodenaufrufe. Wie bereits erwähnt, wird das Modell objektorientiert erfaßt. Was ein Objektmodell von anderen maßgeblich unterscheidet ist das Vorhandensein von Methoden zu diesen Objekten. Nicht alle Methoden müssen an jedem Objekt vorhanden sein. Welche Methoden für welches Objekt zur Verfügung stehen, ist eine Modellierungsfrage. Einige Methoden und Attribute stehen jedoch immer zur Verfügung.

Der Aufruf einer Methode beim ACMI sieht folgendermaßen aus:

```
Protocol: <protocol>
User: <user>
Object: <object>
(Role: <role>)
Method: <method>
(Data: <data>)*
```

Die Zeile “Role” wird nicht bei jeder Anfrage benötigt. Es können je nach Methodenaufruf Daten folgenen. Welche Daten das sind, richtet sich nach dem Methodenaufruf. Im Fall der für das SME wichtigen Methode “Method: resolveAddr” folgt nur die Zeile “Target: <target>”.

Gefolgt wird diese Anfrage durch eine Leerzeile. Das Format kann je nach Eintrag in <protocol> variieren. Es sind verschiedene Protokolle geplant, die gemeinsam nur die erste Zeile “Protocol” haben und dann variieren können. Das SME unterstützt zur Zeit nur das Protokoll “TCACM/long”.

<user> ist der Name des Benutzers, der die Abfrage durchführt. Im Fall des SMS bietet es sich an, einen Benutzer im ACM einzurichten, der den Namen “sms” trägt und die notwendigen Zugriffsrechte auf das Modell besitzt, um mindestens mit den für den Mailversand wichtigen Attributen arbeiten zu können.

<object> beschreibt den Pfad zum Objekt, an das der Methodenaufruf gerichtet ist. Im Falle einer Adreßauflösung, wie sie vom SMS benötigt wird, ist es das Wurzelobjekt (“/”). Es kann jedoch auch nur ein Teilpfad durchsucht werden oder eine Anfrage an genau ein Objekt gerichtet werden.

<method> ist der Name der aufgerufenen Methode. Für die Arbeit des SMS ist hier vor allem die Methode “resolveAddress” wichtig.

<target> übernimmt hier die Aufgabe eines Parameters, der an die Methode übergeben wird. Hier steht im Falle von “resolveAddress”, welche logische E-Mailadresse aufgelöst werden soll.

8.1.2 Wie passen ACMI und die Klasse “Acmi” zusammen?

Wie bereits beschrieben, werden die Befehle an das ACMI und die Antworten gekapselt. Dafür existiert eine Klasse “Acmi”. Für die Benutzung habe ich bislang zwei Methoden implementiert.

Die Methode “request” ist eine allgemeine Anfrage an das ACMI mit den o.g. Parametern, jedoch als Methodenaufruf im C++-Stil.

```
String request(String protocol, String user, String object, String method,
String target);
```

Leider können mit diesem Request nicht alle Methoden aufgerufen werden. Request sollte für eine allgemeinere Anwendbarkeit die folgende Form haben:

```
String request(String protocol, String user, String object, String role,
String method, String paraname[], String paravalue[]);
```

Dabei kann je nach Anfrage “role” ein Leerstring sein. Die String-Felder “paraname” und “paravalue” müssen dabei immer die gleiche Größe haben und könnten auf jweils einen leeren String enden. Dabei müßte “paraname” die Bezeichnung des Parameters beinhalten (z.B. “Target”, “Name”, “Value”, usw.) und “paravalue” den Wert, der an die Methode übergeben werden soll (z.B. “hildmann”, “info”, usw.). Ich möchte auf die allgemeine Form eines Requests an dieser Stelle jedoch nicht genau eingehen. Die Implementierung von “request” in der ersten Form reicht für die benötigten Funktionen des SME aus.

Die andere Methode “resolveAddress” ist ein spezieller Anwendungsfall der “request” Methode. Sie ist wie folgt parametrisiert:

```
String resolveAddr(String user, String target);
```

Die Parameter “protocol”, “method” und “object” sind dabei immer mit “TCACM/long”, “resolveAddr” und “/” belegt.

Zur Zeit wird kein anderes Protokoll als “TCACM/long” von der Klasse “Acmi” unterstützt. Denkbar wäre aber auch eine Unterstützung anderer Protokolle, die vom ACMI definiert sind. Man hätte an dieser Stelle statt dem String “TCACM/long” auch einen Aufzählungstypen definieren und diesen an die Methode “request” übergeben können. Ich habe mich beim Entwurf für den String im Klartext entschieden.

Die Methoden der Klasse liefern jeweils den Antwort-String des ACMI an die aufrufende Funktion. Allerdings unterscheiden sie, ob der Aufruf erfolgreich war oder nicht. Antwortet der ACMI mit einer Fehlermeldung, so wird ein leerer String zurückgegeben. Antwortet der ACMI mit einem “Success”, werden nur die Daten und nicht die “Success”-Meldung zurückgegeben.

Beispiele für eine ACM-Konfiguration gebe ich in „Konfiguration des ACMI“ auf Seite 121.

8.1.3 Wartung des ACM ohne WWW-Interface

Bestandteil der “Aufgabe für die Diplomarbeit” war auch, einen Zugang zum ACM zu schaffen, ohne das System über die WWW-Schnittstelle administrieren zu müssen. Wie, so lautete die zugrundelie-

gende Überlegung, administriere ich ein SME, wenn ich das Zugriffskontrollsystem über WWW nicht benutzen möchte?

Auf diese Frage kann ich drei Antworten geben:

1. Ein ACM läßt sich, wie ich später in dieser Arbeit noch zeigen werde, leicht mit einem gewöhnlichen Editor erstellen. Die Modellierung einer Organisation sollte ohnehin graphisch visualisiert werden. Hat man erst einmal ein Modell auf Papier (oder von mir aus auch im CAD-Programm) erstellt, läßt sich die Struktur einfach in die ACM-Sprache übersetzen und mit einem gewöhnlichen ASCII-Editor eingeben. Ich bin der Überzeugung, daß ein ACM, das nur E-Mailadressen und Schlüssel verwalten soll, noch auf diese Weise handhabbar ist.

2. Bei der Erstellung meiner Diplomarbeit benutzte ich auch das Programm Visio 5.0 für Windows der Firma Visio Corporation. Dieses Programm dient der Visualisierung von Geschäftsprozessen. Neben vielen Vorlagen befindet sich auch eine Vorlage für das Visualisieren von Datenbanken im Lieferumfang. Aus dem Diagramm läßt sich direkt eine Datenbank generieren. Dies ist über eine Visual Basic Schnittstelle in Visio realisiert. Leider reichte die Zeit meiner Diplomarbeit nicht mehr aus. Aber es wäre in jedem Fall möglich, eine Visio Vorlage zu erstellen, die in der Lage ist, aus der erstellten Grafik unmittelbar eine Textdatei zu erstellen, die ein ACM enthält [Visio97]. Dabei ist Visio sicherlich nur eines von vielen Werkzeugen, die in der Lage wären, ACM-Quelltexte zu erstellen.

3. Die naheliegendste Möglichkeit wäre jedoch die Steuerung des ACMI über E-Mail. Im folgenden möchte ich beschreiben, wie eine solche Steuerung realisierbar wäre. Dabei ist die interessanteste Frage sicherlich die der Zugriffskontrolle. Nicht jeder Benutzer darf jede Änderung am ACM vornehmen.

8.1.4 Die Idee vom ACM-Mail-Bot

Ich möchte meinen Entwurf ACM-Mail-Bot nennen; also eine Art Roboter, der Mails lesen kann und diese unter bestimmten Umständen auch in Befehle an das ACMI wandeln kann.

Wie bereits beschrieben, nutzt die WWW-Schnittstelle des ACM auch nur das Protokoll, das der ACMI zur Verfügung stellt. Möchte ich also die gleiche Funktionalität zur Verfügung stellen, wie die WWW-Schnittstelle, so muß ich nur in der Lage sein, protokollgerecht Anfragen an das ACMI zu senden.

Genau das macht schon die Klasse “Acmi”. Diese Klasse enthält die Methode “request”, mit der Befehle an das ACMI gesendet werden können. Die Antworten kommen dann als Strings an die aufrufende Funktion zurück.

Das einzige, was an einem ACM-Mail-Bot also noch fehlt, ist ein Parser, der Anfragen in einer zu definierenden Form zerlegt und diese mittels Acmi.request()-Methode verschickt. Die Antwort des ACMI kann mit einigen Zusatzinformationen angereichert werden und schließlich an den Absender zurückgesandt werden.

Ich bin kein Freund vom “neu Erfinden des Rades”. Aus diesem Grund würde ich folgende Syntax für das Absenden von ACMI-Anfragen bevorzugen:

```
Object: <object>
Method: <method>
Role: <role>
Data: <data>...
```

Dabei könnte “Data: <data>” hier wieder ein Parameter, wie z.B. “Target: <target>” sein und es könnten je nach Methode noch weitere Parameter folgen.

Alternativ wäre folgende Syntax denkbar:

```
<object>.<role>.<method>(<data>, ...)
```

Letzterer Vorschlag würde selbst in einer Subject-Zeile Platz finden. Da die Header jedoch unverschlüsselt über das Netz gehen, halte ich die Plazierung im Header für sehr ungeschickt. Ich würde die Übertragung im Mail-Body vorziehen.

Wie könnte nun das Versenden einer Nachricht an einen ACM-Mail-Bot aussehen?

Ein Administrator (nennen wir ihn “Hildmann”) möchte einen Benutzer löschen. Also versendet er eine E-Mail, die so aussehen könnte:

```
From: hildmann@prz.tu-berlin.de
To: acm-mail-bot@tc.tu-berlin.de

BEGIN
  Object: /
  Method: destroyObj
  Name:   mueller14
END
```

Diese E-Mail wird nun über das SME verschickt. Das bedeutet, daß sie von den SPT-Tools auf dem Rechner von Hildmann verschlüsselt und signiert wird. So wird sie an den Rechner “tc.tu-berlin.de” geschickt (TC steht hier für Trust Center und ist im Grunde ein beliebiger Name) und auf dem Weg vom SMS ggf. umverschlüsselt.

Angenommen der ACM-Mail-Bot ist im ACM genauso als E-Mail-Empfänger eingetragen wie ein normaler Benutzer, dann würde das SMS oder der SMG, hinter dem der ACM-Mail-Bot plaziert ist, seine Mail zugestellt bekommen und könnte sich auf Grund der Sicherheitsmechanismen im SMS relativ sicher sein, daß sowohl der Absender authentisch ist, wie auch der Mailtext.

Lassen wir nun den SMTP-Server die Mail nicht ausliefern, sondern tragen wir stattdessen ein kleines Programm als sog. Alias ein. Bei Sendmail sähe das ungefähr so aus:

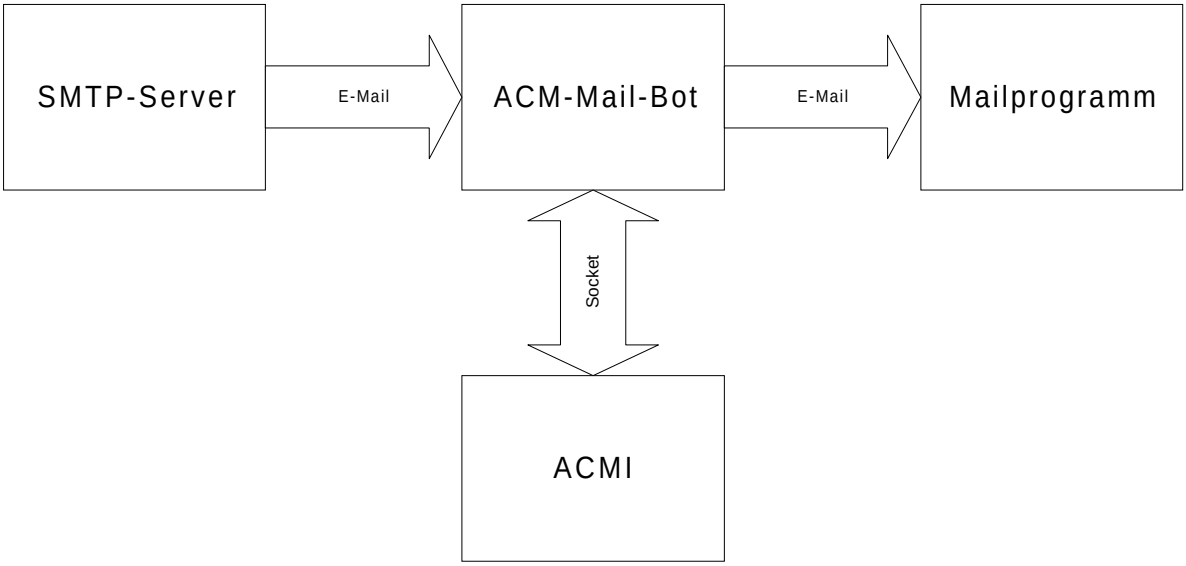
```
# Datei: /etc/mail/aliases

# ... andere Aliaseinträge ...
acm-mail-bot:           “|/usr/local/bin/acm-mail-bot”
# ...
```

Das würde bedeuten, daß statt einer Auslieferung in einen Posteingangsordner das Programm “/usr/local/bin/acm-mail-bot” gestartet werden würde.

Dieses Programm würde dann die oben abgebildete E-Mail als Standardeingabe bekommen. Dank der SME Bibliothek wäre das Einlesen der E-Mail über die Klasse Mail ebenso kein Problem, wie die

Verbindung zum ACMI. Alles, was noch implementiert werden müßte, wäre ein Parser, der die Zeilen aus dem Body der Mail einlesen kann.

ABBILDUNG 56. Architektur: ACM-Mail-Bot



Aber für den Aufruf von Acmi.request() fehlt noch der Parameter “user”. Diesen könnten wir bequem aus der From-Zeile des Headers gewinnen. Es müßte jedoch vorher geprüft werden, ob die Mail auch wirklich aus der dem SME angehörigen Domain stammt und die Mail wirklich unterschrieben ihr Ziel erreicht hat. Zu diesem Zweck könnte der ACM-Mail-Bot sicherheitshalber noch einmal die Header-Signatur der E-Mail überprüfen. Selbst, wenn die Mail vom SMG bereits geprüft und dekodiert wurde, bleibt die Zeile X-Signature noch erhalten und kann jederzeit erneut geprüft werden. Ob der Body noch entschlüsselt und geprüft werden muß, hängt von der Frage ab, ob das Programm hinter einem SMG läuft oder auf dem SMS Rechner. In jedem Fall stehen die Funktionen zur Verfügung.

Ist der Mailtext ausgewertet und der Absender ermittelt und durch die Header-Signatur authentifiziert, kann die Anfrage an das ACMI gestartet werden. Für die Antwort kann der ACM-Mail-Bot wiederum eine neue E-Mail zusammenstellen und den Antwort-String des ACMI an den Absender zurückschikken. Dies geschieht auch wieder verschlüsselt und signiert.

Der Parser könnte ferner so beschaffen sein, daß er eine beliebig lange Sequenz von Anfragen lesen könnte. Die Antworten könnten dann gesammelt werden und wieder als eine Antwortmail zurückgesandt werden.

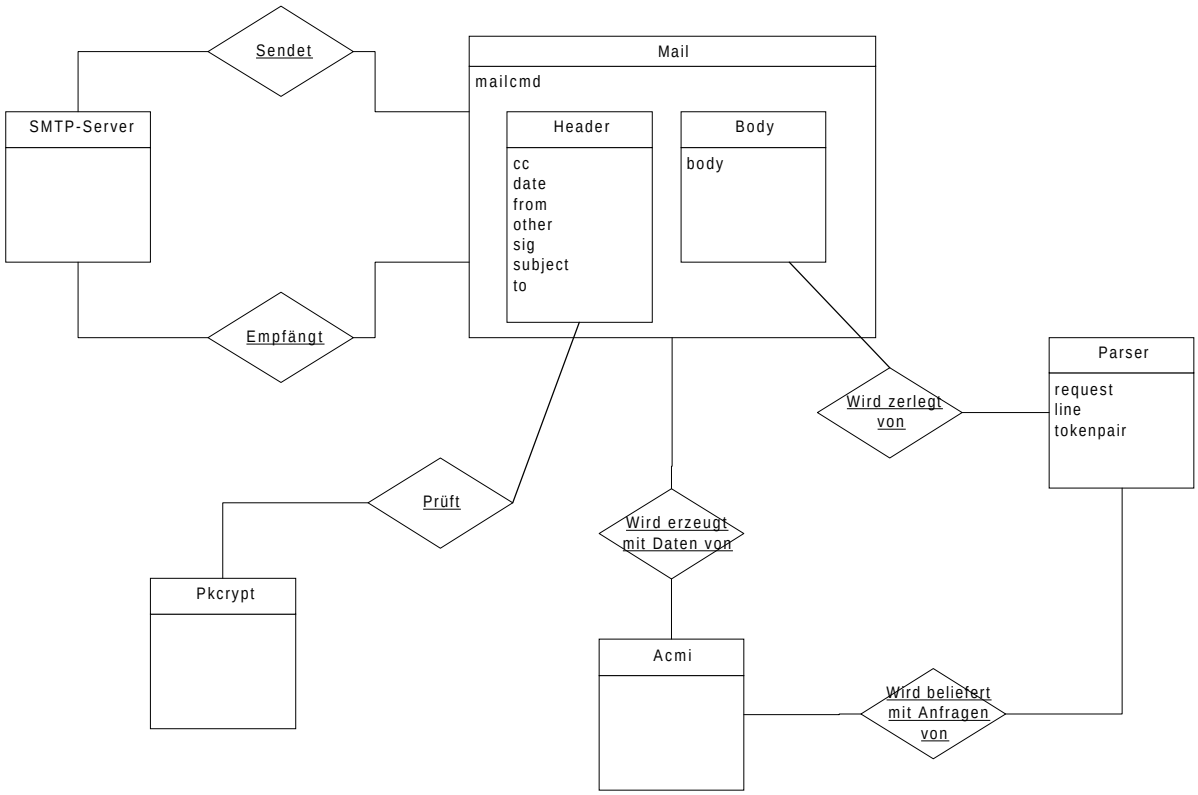
Das würde sogar eine relativ komfortable asynchrone Wartung ermöglichen. So könnte der Administrator in einer Mail einen Satz von Anfragen fertigstellen und müßte nicht einmal eine Verbindung zum Netzwerk haben. Beim Absenden der E-Mail müßte kurzfristig eine Verbindung hergestellt werden. Ebenso beim Abfragen der Antworten.

Störend könnte an dieser Stelle sein, daß der ACMI keine Rollbacks kennt, wie das bei Datenbanken üblich ist. Es könnte so z.B. sinnvoll sein, eine Sequenz von Anfragen zu einer Transaktion zusammenzufassen, diese Transaktion aber als ganzes nur dann auszuführen, wenn auch alle Anfragen ohne Fehler beantwortet werden würden und ansonsten wieder in den Ausgangszustand zurückzukehren, also bevor diese Transaktion begonnen hat. Dies sei nur als Anregung für eine Weiterentwicklung des ACMI erwähnt.

8.1.5 Analyse des ACM-Mail-Bot

Das folgende Objektmodell zeigt die Beziehungen der beteiligten Objekte:

ABBILDUNG 57. Objektmodell: ACM-Mail-Bot



Ich gehe in dieser Darstellung davon aus, daß das Programm bereits entschlüsselte Daten erhält, also z.B. hinter einem SMG angesiedelt ist. Im anderen Fall genügt es, die entsprechende Entschlüsselungsroutine von Cryptedmail aufzurufen. Damit entstünde aber eine weitere Beziehung im oben abgebildeten Modell.

Alle Objekte sind bereits bekannt und in der SME Bibliothek enthalten. Die Ausnahme bildet die Klasse “Parser”.

Ich definiere folgende kontextfreie Sprache nach Backus und Naur (bekannt als Erweiterte Backus-Naur-Form, EBNF):

```
ausdruck::= ’begin’ {anfrage}* ’end’
anfrage::= {zeile}* ’\n’
zeile::= name ’: ’ inhalt ’\n’
name::= ’object’|’method’|...
inhalt::= alpha {sonder|alpha|ziffer}*
sonder::= '!'|'"'|'/'|...
alpha::= 'a'|'b'|'c'|...’A’|’B’|’C’|...
ziffer::= '1'|'2'|'3'|...
```

Bei der Auswertung der Schlüsselwörter finden Groß- und Kleinschreibung keine Beachtung. Die “inhalt”-Tokens werden unverändert an das ACMI weitergegeben. Die Zeichensequenz `'\n'` steht für ein Zeilenende.

Der Parser läßt sich in drei Stufen aufbauen. In der ersten Stufe wird der Mailbody in Anfragen zerlegt, im zweiten Schritt werden die Anfragen in Zeilen zerlegt und im dritten Schritt die Zeilen in ein Paar (Name, Inhalt). Üblicherweise wird intern ein Baum aufgebaut, der eine Repräsentation des Ausdrucks bildet.

Da die Sprache extrem einfach gehalten ist, ließe sich eine einzige Operation definieren, die die drei Parameter für die Anfrage beim ACMI liefern könnte.

ABBILDUNG 58. Operationsmodell: Parser.nächsteAnfrage

Operation:	nächsteAnfrage
Beschreibung:	Liefert die nächste komplette Anfrage aus dem gelesenen Ausdruck oder eine Fehlerbeschreibung.
Nur Lesen:	syntaxbaum : btree
Ändern:	SUPPLIED {method, object, target, error} : String, syntaxbaum : btree
Senden:	-
Angenommen:	Aus einem gegebenen Ausdruck wurde bereits der Syntaxbaum erstellt.
Ergebnis:	Liegt eine syntaktisch korrekte Anfrage vor, sind method, target und object sonst error belegt. Der Syntaxbaum wird abgebaut.

Eine weitere Operation gibt Auskunft über den Zustand des Syntaxbaums und dient als Abbruchkriterium.

8.2 Der Keyhandler

Im Kapitel „Automatisierung von sicherem elektronischen Schriftverkehr“ auf Seite 25 hatte ich schon den Ansatz von Michael Herfert des “seperation of duty” erwähnt. In der Entwicklungs- und Implementierungsphase bin ich jedoch nicht weiter darauf eingegangen. Der Grund: Es fehlte mir gegen Ende der Arbeit einfach die Zeit, mich mit der praktischen Umsetzung zu befassen. Grund hierfür sind Probleme bei der Portierung gewesen. Ich möchte hier jedoch darstellen, wie eine solche Implementierung aussehen würde und damit zeigen, daß der nächste Schritt nicht mehr groß, wenn auch sicherlich nicht zu unterschätzen ist.

Die SME-Bibliothek enthält eine Klasse, die sich Pkcrypt nennt. Diese Klasse beinhaltet ein aus meiner Sicht sehr allgemeines Modell von kryptographischen Algorithmen, die auf Public-Key Verschlüsselungsverfahren aufsetzen. Diese Klasse enthält die Methoden:

sign(), mkxsig(), unmkxsig(), crypt(), decrypt() und check() sowie diverse Methoden, die eine gewisse Bedeutung für das Programmieren mit dieser Klasse haben.

Michael Herfert entwickelte bereits einen Prototypen eines Keyhandlers, der zumindest in der Lage ist, PEM verschlüsselte E-Mails umzuverschlüsseln. Das Vorgehen dabei ist das folgende:

- Extrahieren des mit dem öffentlichen Schlüssel des Empfängers verschlüsselten symmetrischen Schlüssels (cs.crypt(pk1)). Dieser wird bei PEM mit zwischen die markierten Zeilen geschrieben und ist leicht erkennbar. Für diesen Vorgang gibt es bereits eine Beispielimplementierung.
- Dieser Schlüsselkomplex wird dann mittels Socketverbindung über ein definiertes Protokoll inklusive der gewünschten Empfängeradressen zum Keyserver übertragen.
- Der Keyserver extrahiert den symmetrischen Schlüssel und verschlüsselt diesen neu mit der gewünschten Adresse (cs.crypt(pk2)).
- Das Ergebnis wird wieder über die Socketverbindung zurückgesendet.

Für die Integration des Konzeptes in das SMS gibt es verschiedene Optionen, die ich nan dieser Stelle kurz diskutieren möchte:

Es ist bereits so, daß es zwei Klassen gibt, die von Pkcrypt abgeleitet sind. Diese Klassen heißen Pkcryptpgp und Pkcryptpem. Diese implementieren jeweils die gleichen Funktionalitäten auf Basis verschiedener Verschlüsselungsstandards. In den Hauptprogrammen wird jedoch jeweils mit Pkcrypt gearbeitet, welches dann die benötigte abgeleitete Methode aufruft. Welche Ableitung jeweils richtig ist, entscheidet Pkcrypt z.Zt. lediglich über den Aufbau der Mailadresse. PEM Identifizierer sind nach X.500 Syntax aufgebaut, PGP Identifizierer sind nach Arpa-Standard aufgebaut.

Eine mögliche Implementierung könnte eine weitere Vererbung von Pkcrypt initiieren, die dann Pkcryptkh heißen könnte. Diese Implementierung wäre fast identisch mit Pkcryptpem, nur daß die Methoden sign(), check(), crypt() und decrypt() angepaßt werden müßten. Statt die PEM-Routinen aus der Secude Bibliothek aufzurufen, müßten hier Programmteile nach dem Vorbild von Michael Herfert untergebracht werden. mkxsig() und unmkxsig() sind im Grunde nur Routinen, die das Format der Signaturausgabe ändern, um sie der Syntax von RFC 822 Mailheadern anzupassen. Da die Ausgaben von Michael Herfert ebenso PEM-Ausgaben sind, wie die der Secude Bibliothek, müßten nur die Aufrufe der Bibliothek durch die Socketkommunikation mit dem Keyhandler ersetzt werden.

Da sich die Schlüssel-Identifizierer zwischen Pkcryptpem und Pkcryptkh nicht mehr unterscheiden, könnte man hier zur Unterscheidung das Attribut Pkcrypt.pkpath benutzen. Beinhaltet Pkcrypt.pkpath einen gültigen Pfad zu einem Verzeichnis, wird davon ausgegangen, daß es sich um eine Software-PSE handelt. Eine PSE ist ein Konstrukt der Secude Bibliothek und enthält sowohl öffentliche als auch private Schlüssel. Steht in Pkcrypt.pkpath jedoch ein Rechnername und eine Portnummer in der Form “hostname.domain:port” also z.B. “hoogla.prz.tu-berlin.de:90210”, wird auf die Methoden von Pkcryptkh zugegriffen und der Rechnername und die Portnummer für das Aufbauen der Verbindung zum Keyhandler benutzt. Dieser Ansatz paßt durchaus in das Konzept, das von Secude vorgegeben wurde. Auch hier ist es so, daß zur Identifizierung eines Smartcard-Lesegerätes an Stelle einer Software-PSE als Pfad das Gerät angegeben wird (z.B. “gpk2000”).

Die Anpassungen, die dann im SMS gemacht werden müssen, sind lediglich Konfigurationsarbeit. Hier muß der PSE-Pfad durch Rechnernamen und Portnummer in o.g. Form ersetzt werden. Selbstverständlich muß die neue Version der Bibliothek an das Hauptprogramm gelinkt werden.

Ein anderer Ansatz wäre, Pkcryptpem so zu ändern, daß hier die Unterscheidung nach Pfad oder Netzwerkverbindung gemacht würde. Nach außen wäre die Benutzung die gleiche. Es würde nicht Pkcrypt geändert werden, sondern Pkcryptpem. Da es sich bei Pkcryptpem um eine getestete Klasse handelt, würde ich jedoch von Änderungen hier absehen. Die Klasse Pkcrypt ist sehr viel kleiner und übersichtlicher. Hier können Änderungen schnell überprüft und ggf. rückgängig gemacht werden. Die Klasse Pkcryptkh könnte ferner zunächst für sich in einem eigenen Programm getestet werden, bevor sie im SMS zum Einsatz kommt.

Ein gänzlich anderer Ansatz wäre, die Keyhandlerfunktionen aus der Bibliothek zu ziehen und direkt im SMS zu implementieren. Diese Idee halte ich deshalb für schlecht, weil der SMS wiederum getestet ist und für sich so erst einmal läuft. Hier sollten wirklich nur echte funktionale Erweiterungen gemacht werden. Vielleicht macht es ja auch an irgendeiner Stelle Sinn, z.B. auch im SMG einen Keyhandler einzusetzen. Insofern plädiere ich in jedem Fall für eine Lösung, die in der Bibliothek implementiert wird.

8.3 LDAP, S/MIME, Verzeichnisdienste

Wie ich mit dem Beispiel Keyhandler deutlich gemacht habe, ist die Klasse Pkcrypt der ideale Ort zur Erweiterung des SME. Möchte man z.B. die Benutzung von S/MIME konformen Mails als Möglichkeit hinzufügen, so muß auch hier nur eine weitere Pkcryptsmime Klasse entwickelt und dem bestehenden Pkcrypt hinzugefügt werden.

Ähnlich sieht es auch mit Verzeichnisdiensten und Keyservern wie LDAP aus. Wie schon zu Beginn dieser Arbeit erwähnt, speichern Keyserver die öffentlichen Schlüssel von beliebigen E-Mail-Nutzern. Eine Erweiterung der Pkcrypt-Klasse um ein Pkcryptldap könnte beispielsweise auch eine Anfrage an einen LDAP-Keyserver zulassen. Der Weiterentwicklung sind hier keine Grenzen gesetzt. Nur ist es noch einmal ein ganz erheblicher Entwicklungsaufwand, eine Klasse zu erstellen, die eine protokollkonforme Anfrage bei einem Keyserver stellt, daraufhin die Signatur des Verzeichnisdienstes prüft und den so gewonnenen öffentlichen Schlüssel des Empfängers einer kryptographischen Routine zugänglich machen kann.

8.4 Virusschutz

Im Bereich EDV-Sicherheit hat ein regelrechtes Wettrüsten begonnen. Nur habe ich in bestimmten Bereichen das Gefühl, daß hier nicht Cracker und Sicherheitsexperten gegeneinander aufrüsten, sondern Sicherheitsexperten gegen Sicherheitsexperten. Bestimmte Sicherheitsprodukte bieten Sicherheitsmechanismen an, die nach meinem Dafürhalten an dieser Stelle gar keinen rechten Sinn ergeben.

So kann man sich über den Sinn oder Unsinn eines Virusschutzes in E-Mail Systemen streiten. Bei der Entwicklung des SME ist jedoch die Frage nach einem Virusschutz aufgetaucht, und so möchte ich die zur Verfügung stehenden Möglichkeiten an dieser Stelle diskutieren.

Klaus Brunnstein machte sich einen Namen als der “Viren-Professor”. Er meint am 19. Oktober 1997 in einer E-Mail im Rahmen einer öffentlichen Mailingliste auf eine Anfrage bezüglich der Verläßlichkeit von UNIX-Virenscannern:

Sehr geehrte Frau Xxxxx, einige Hersteller von AntiViren-Software arbeiten zwar an UNIX-On-Access-Scannern, aber die bisherigen Produkte haben noch nicht einen Stand, den man zum Einsatz empfehlen kann. Selbst die W-NT und W95-Scanner-Ergebnisse bleiben zT deutlich in Erkennung und Qualitaet hinter den DOS/W3.x Produkten zurueck.

Er gibt bei dieser Gelegenheit auch folgende URL an:

`http://agn-www.informatik.uni-hamburg.de/vtc.`

Leider befinden sich bis heute keine Vergleichsergebnisse zwischen UNIX-Virenscannern und DOS-Virenscannern auf diesen Seiten (Stand: 5. November 1998).

Da laut Anforderung das SME auf den im Einsatz befindlichen UNIX-Maschinen laufen soll, gibt es laut Aussage von Herrn Brunnstein keine “empfehlenswerten” Programme. Dies kann sich jedoch mittlerweile geändert haben.

Eine E-Mail kann für sich zunächst einmal keine Viren enthalten. Es handelt sich um einen einfachen ASCII-Text, der nicht interpretiert wird. Erst durch “unvorsichtig” implementierte E-Mail Clients und MIME-Mails kann überhaupt eine Gefahr bestehen. Ein Fall, der mir bislang bekannt ist, bei dem eine E-Mail gefährlich werden kann, ist die Verwendung von Word als E-Mail-Anzeige-Programm von Outlook oder Exchange aus. Meines Wissens nach wird der Benutzer bei den aktuellen Versionen jedoch immer darauf aufmerksam gemacht, daß es jetzt darum geht, eine Word-Datei zu öffnen. Und wenn mich nicht alles täuscht, tauchen mittlerweile auch Meldungen auf, die den Benutzer darauf hinweisen, daß es sich um einen sog. aktiven Inhalt handelt (die Microsoft-Umschreibung für Macros), und daß diese Inhalte auch Viren enthalten können (sog. Macro-Viren).

Also fassen wir zusammen:

- einfache E-Mails können keine Viren enthalten
- MIME-Attachments, als an E-Mails angehängte Dateien können Viren enthalten. Diese Inhalte werden jedoch nicht automatisch aktiv.

Der Benutzer hat also die Gelegenheit, vor dem Aktivieren von “aktiven Inhalten”, eine Virusprüfung durchzuführen. Ferner gibt es Produkte, die als Erweiterung von Word angeboten werden, die von Word aus Macro-Viren vorbeugen sollen. Applikationen können mittels Virenscanner geprüft werden.

Der Schutz vor Viren könnte auf zwei Arten geregelt werden. Der ganz pragmatische Ansatz wäre der, das Öffnen von Attachments nur dann zu genehmigen, wenn diese Attachment von einer EDV-Stelle stammen, die selbst die Virenprüfung vorgenommen hat. Experten könnten hier alle versendeten und empfangenen Dateien prüfen und weiterverteilen. Durch die Signierung der E-Mail ist auch sichergestellt, daß auf dem Weg von dieser Anti-Virus-Stelle zum Empfänger kein Virus mehr hinzugekommen ist.

Der andere Ansatz wäre, die Installation von Anti-Viren-Programmen auf allen PCs, die ggf. Attachments verarbeiten können. Dies wirft jedoch wieder die Probleme der Verteilung der jeweils neuesten Version und die enormen Kosten für ggf. tausende von Anti-Viren-Lizenzen auf.

Ein Kompromiß wäre an dieser Stelle, am Tag X alle am SME beteiligten Rechner nach Viren geprüft zu haben und zu untersagen, daß irgendwelche Programme oder Macros von außen dem System hinzugefügt werden. Gesetzt den Fall, dies würde tatsächlich funktionieren (je kleiner der Kreis der Teilnehmer ist, desto wahrscheinlicher ist dies), könnten die SME-Teilnehmerinnen und Teilnehmer untereinander bedenkenlos Software und Macros austauschen, ohne Gefahr zu laufen, sich irgendwo zu infizieren.

Was aber, wenn eine Überprüfung durch das SME praktisch als Application-Level-Firewall-Applikation gefordert ist. Eine sehr zentrale Stelle für die Überprüfung von Mails wäre das SMS. Das Überprüfen von Viren würde hier jedoch zur Folge haben, daß die E-Mails doch entschlüsselt werden müssen und das “seperation-of-duty”-Konzept hier nicht greifen könnte. Eine andere Stelle wären die SMGs, die, wie wir ja wissen, sowieso Klartext erzeugen und die Mails unverschlüsselt zum Abholen bereitstellen. Mit SPTs ausgestattete Rechner müßten selbst für ihren Virusschutz Sorge tragen.

Der Viruswächter selbst wäre wiederum ein Filter, der sich beliebig zwischen die Abläufe im SME dazwischenschalten ließe. Seine Aufgabe wäre:

- Extrahiere alle Attachments aus der E-Mail.
- Für jedes Attachment befolge folgende Anweisungen:

- Handelt es sich bei dem Attachment um eine gepackte Datei, packe sie aus und reihe sie in dieser Form in die Liste der Attachments ein.
- Handelt es sich um eine ungepackte Datei, prüfe, ob sie aktive Inhalte enthält.
- Enthält sie aktive Inhalte, führe sie einem externen Virenwächterprogramm zu.
- Sind alle Inhalte “sauber”, sende die E-Mail weiter, sonst erzeuge eine Fehlermeldung.

Wie so oft liegt die Tücke hier im Detail. Die unscheinbare Anweisung: “Handelt es sich bei dem Attachment um eine gepackte Datei, packe sie aus und reihe sie in dieser Form in die Liste der Attachments ein.”, wird sich zur Monsterfunktion auswachsen, da hier eine gewaltige Zahl an Packprogrammen registriert werden muß und zunächst analysiert werden müßte, um welches Komprimierungsformat es sich handelt. Der einfachste Lösungsansatz wäre hier, von der Dateiendung auf den Inhalt zu schließen und über eine MIME-Typenliste auf das Packprogramm zu schließen, das in der Lage ist, diesen Job zu übernehmen.

Als externes Programm kommt meines Wissens eine Linux-Variante des McAffee Anti-Virus-Tools in Frage. Diese Software habe ich bereits gesehen aber noch nicht ausgiebig getestet. Im Paket enthalten sind Binaries für die ELF- und die AOUT-Architektur. Das Programm “uvscan” wird mit der zu prüfenden DOS/Windows-Datei aufgerufen und meldet entweder einen Fehler oder im Normalfall nichts. Ich habe dieses Programm noch nicht ausgiebig testen können.

In jedem Fall könnte der Virenprüfmechanismus relativ leicht mit in die Programme SMS und SMG eingebaut werden.

Eine weitere Alternative wäre die Nutzung eines externen DOS-Programms zur Überprüfung. Dafür gäbe es zwei Ansätze:

1. Man findet ein DOS-Virenwächter-Programm, daß im DOS-Emulator von Linux läuft und findet einen Protokollablauf, der die Interaktion zwischen dem Linux-Programm und dem DOS-Programm gewährleistet (vermutlich über Zustände in Dateien).

2. Der Virenwächter schickt die unverschlüsselte E-Mail an einen externen Viren-Schutz-Rechner, der unter DOS läuft (oder unter Windows) und erwartet ein “O.K.” von diesem Rechner zurück.

Da nach Aussage des “Viren-Professors” die UNIX-Virenscanner zur Zeit nicht empfehlenswert sind, würde ich vorschlagen, vorerst auf eine Prüfung unter DOS zurückzugreifen. Dabei kann überlegt werden, ob die Prüfung auf den jeweiligen Arbeitsstationen durchgeführt werden soll oder an einer zentralen Stelle, die jede E-Mail passieren muß. Es ist jedoch zu bedenken, daß sich in E-Mails für UNIX-Systeme nach heutigem Wissensstand gar keine Viren befinden können, somit also eine Prüfung dieser E-Mails unnötig ist.

8.4.1 Nachtrag

In der Mailinglist: win-sec@cert.dfn.de fand Anfang November eine Diskussion über Sicherheitslükken in Microsoft Mailprogrammen statt, die das Injizieren von E-Mailviren ermöglichen würden. Hierbei wurde auf das "Microsoft Security Bulletin (MS98-008)" hingewiesen.

Die von mir erwähnte “unvorsichtige” Implementierung eines E-Mail Clients hat also bereits stattgefunden. Darauf kann auf zwei Arten reagiert werden. Die eine Möglichkeit wäre, Outlook aus Sicherheitsgründen nicht weiter zu verwenden. Die andere Möglichkeit ist, sich die korrigierte Fassung von Microsoft zu beschaffen und zu hoffen, daß diese keine weiteren Lücken aufweist.

KAPITEL 9

Implementierung

9.1 Struktur der Implementierung

Objektorientierte Entwicklung befreit einen Entwickler weder von der Notwendigkeit einer durchdachten Modularisierung, noch von einer sinnvollen Planung, wann welche Module fertiggestellt werden und wie sie getestet werden können.

Das SME in seiner jetzigen prototypischen Implementierung besteht aus ca. 30 Dateien mit ca. 4500 Zeilen Programmtext. Interessant ist, daß zwischenzeitlich die Zeilenzahl erheblich geschrumpft ist, nachdem einige Prototypen durch vollständig neue Versionen ersetzt wurden.

Ich bin ein Feind von unübersichtlichen, gewachsenen Programmtexten, die Fehler aus Urzeiten in die Gegenwart transportieren. Softwareentwickler sollten Menschen der Gegenwart sein. Nostalgie hat nichts im Quelltext eines Programms zu suchen. Leider sind diese Ansichten nicht an jeder Stelle sehr beliebt, und das aus den folgenden Gründen:

Wie bereits ganz am Anfang dieser Arbeit erwähnt, handelt es sich beim SME um ein Konzept, das Schritt für Schritt durch die Entwicklung immer neuer Prototypen entstanden ist, dessen großer Verdienst zum Teil überhaupt erst die Erstellung einer Anforderungsliste war, die auf andere Arten kaum zu beschaffen war.

Es ist in Softwareprojekten eher die Regel als die Ausnahme, daß vor der eigentlichen Systemerstellung eine Phase von Testprogrammen und Prototypen stattfindet. Und immer wieder wird aus einem Prototyp letztendlich ein endgültiger Prototyp gemacht. Was ich damit sagen will ist, daß der als Prototyp geplante Programmtext eines Tages zum Produkt gekrönt wird. Die Gründe sind einfach und lassen sich bei unserem Projekt genauso erkennen, wie ich das bei anderen Projekten miterlebt habe oder auch von anderen gehört habe.

9.1.1 Die Geschichte des SME

Als ich zum E2S Projekt kam, ersetzte ich einen ehemaligen Mitarbeiter aus dem Rechnerbetrieb des PRZ der TU Berlin, der seit einiger Zeit in diesem Projektkontext arbeitete und ein wahrer Künstler im Umgang mit dem SMTP-Server “Sendmail” war. Ferner gab es einen “Urgroßvater” des SMG, genannt Mailexploder (MEX). Der MEX basierte auf einer Anzahl von Shellskripten, einer Sendmail-Konfiguration, einiger Procmail-Skripte und Anleitungen zur Konfiguration einer Firewall. Das System arbeitete auf der Basis von PGP.

Nach einiger Zeit im Projektkontext und nachdem ich die Papiere gelesen hatte, die bereits zu diesem Thema verfaßt wurden, begann ich mit der Entwicklung der Konzeption von SMS und SMG. Diese Konzeption war nicht zwingend erforderlich. Aber sie war die in meinen Augen einzige Lösung, die auch mit den Ideen von Michael Herfert vereinbar war. Ich konnte mir gut vorstellen, von jedem SMG zum ACMI eine gesicherte Verbindung aufzubauen, um die Namen aufzulösen. Aber den Schlüsseltransport auch darüber zu organisieren oder wahlweise alle MEX-Rechner durch CMW-Maschinen auszutauschen, erschien mir im Kontext der TU doch sehr hoch gegriffen. Immerhin war unser Projektziel auch, eine Variante zu schaffen, die man in der TU Verwaltung einsetzen kann.

So kam ich zu diesem Zeitpunkt gar nicht in Versuchung, die alten Quelltexte anzurühren und für das Konzept SMS / SMG zu nutzen. Aus dem Projektkontext ergab sich, daß von einer PGP-Verschlüsselung zur PEM-Verschlüsselung gewechselt wurde. Zum einen wurde der Wechsel nötig, um den Ideen von Michael Herfert gerecht zu werden und zum andern, um die Verwendung von Smartcards zu ermöglichen.

Ich wollte in jedem Fall eine C++-Implementierung durchführen. Warum? Nun, als Student hat man die Gelegenheit, Dinge zu lernen, zu denen man später keine Zeit mehr hat. Ich hatte in der Basisveranstaltung “Softwaretechnik und Systemgestaltung” die Fusion Methode kennengelernt und in einem Seminar “Objektorientierte Systementwicklung” vom goldenen Zeitalter der objektorientierten Entwicklung von Software gehört. In einigen Projekten hatte ich bereits die Gelegenheit, mit Java zu arbeiten, und nicht objektorientierte Sprachen, wie C, Modula2 oder TCL/TK waren bekannt und damit für mich uninteressant. Java wäre gerade wegen seiner Plattformunabhängigkeit eine Alternative gewesen. Für die Entwicklung einer sog. native C-Schnittstelle zu Java war jedoch keine Zeit, so daß ich mich aus diesen Überlegungen heraus für C++ entschied. Das Wichtigste jedoch war: Ich wollte C++ lernen, und es gab eine C-Schnittstelle zu PEM über die Bibliothek Secude, die ich brauchte. Ich mochte damals C nicht und kann es noch immer nicht leiden, und C++ war wenigstens eine Sprache, mit der alle Projektpartner etwas anfangen konnten. Die Erstellung einer Schnittstelle zwischen Java und Secude hätte mich weit ab von meinen wirklichen Problemen gebracht. Außerdem kann eine Native-Schnittstelle, wie die von Secude zu Java immer nur auf einer Plattform betrieben werden. Das heißt, daß in diesem Fall Java seine Plattformunabhängigkeit verloren hätte.

Die Secude-Bibliothek ist zu etwas mehr in der Lage, als nur die Unterstützung von PEM. Das ist gut so, und das ist schlecht so. Denn die Dokumentation ist sehr umfangreich, und es gibt keine leicht verständlichen Tutorials oder Fremdliteratur zu diesem Thema. Also legte ich das Problem Secude zunächst beiseite und widmete mich dem Kernproblem. Ich implementierte also die Klassen Mail und Pkcrypt, so wie beschrieben. Pkcrypt hatte natürlich zu diesem Zeitpunkt nicht das Strategie-Muster-Konzept, nach dem anhand eines Entscheidungskriteriums jeweils der richtige Algorithmus ausgewählt wird [Gam96], sondern war eigentlich dazu gedacht, im Quelltext ersetzt zu werden.

Der nächste Schritt war die Integration des ACM in das Konzept und die Entwicklung der Header-Signatur. Für jede Klasse, die ich erstellte, entwickelte ich auch gleich eine Testumgebung, um sie unabhängig vom Gesamtkonzept testen zu können. Das kostete natürlich alles Zeit, auch wenn ich noch heute davon überzeugt bin, daß eine Fehlersuche im Hauptprogramm mindestens genauso lange gedauert hätte. Dann war der Zeitpunkt des nächsten Meilensteins schon wieder erreicht.

Fast keine Zeit blieb für eine anständige Konfiguration und für eine Demo mußte auch ganz schnell die Lösung für alleinstehende PCs her. Ich brauchte also schnellstens eine Lösung für DOS, die grundsätzlich die gleichen Konzepte verfolgte, wie die UNIX-Version. Ich besorgte mir also den GNU C-Compiler für DOS und “hackte” in ein paar Tagen und Nächten etwas wie zwei Proxyprogramme zusammen.

Die Prototypen liefen (irgendwie). Sie hatten nur den Nachteil, daß man pro E-Mail drei Dialogboxen wegklicken mußte, und das sowohl beim Senden, als auch beim Empfangen von E-Mails. Diese Unannehmlichkeit war darin begründet, daß unter Windows bei jedem Start eines Programms der Secude-Suite eine Copyrightmeldung auf dem Bildschirm erschien.

Der Odyssee der Windows-Portierung widme ich ein eigenes Unterkapitel. Dieser Abschnitt der Arbeit würde sonst nicht gebührend zur Geltung kommen.

Im letzten Abschnitt der Arbeit half mir mein damaliger Kollege Ehab Jewabreh bei der Anbindung der Secude-Bibliothek an die Pkcrypt-Klasse. Somit stehen nun für die Pkrypt-Klasse nicht nur zwei Algorithmen zur Verfügung, sondern im Grunde auch zwei Implementierungen des gleichen Algorithmus. Eine basierend auf externen Programmaufrufen, die andere basierend auf API-Aufrufen der Secude-Bibliothek.

Walter F. Tichy macht in [Tich79] die Einteilung in Mitglieder einer Modulfamilie. Er spricht von “implementations”, “revisions” und “derived versions”. Von Pkcrypt gibt es nicht nur verschiedene “revisions”, also Entwicklungsstände, sondern auch zwei “implementations”, also verschiedene Implementierungen mit der gleichen Operations-Schnittstelle nach außen. Über ein Compiler-Makro kann man zwischen den beiden Implementierungen wählen.

Leider ist die Anbindung über die Secude-Bibliothek bis heute noch nicht einwandfrei, weil noch mit statischen Feldgrößen gearbeitet wird. Eine wichtige Arbeit in der Weiterentwicklung des SME wäre also das Zwischenstück zwischen Pkcryptpem und der Bibliothek, bestehend aus vier C-Funktionen so umzuschreiben, daß mit dynamischen Größen gearbeitet werden kann.

Bevor ich mich der Probleme “seperation-of-duty” und dem ACM-Mail-Bot widmen wollte, stand für mich eine Neuimplementierung (engl. rewrite) der wichtigsten Funktionen auf dem Plan. Das heißt, ich wollte die Hauptprogramme umschreiben und die Klassen überarbeiten. So hatte ich einige Funktionen, wie das Weiterleiten der E-Mail sowohl im SMS als auch im SMG implementiert. Diese Implementierung rückte nun in die Klasse Mail. Ich nutzte die Gelegenheit und führte nicht nur meine Nachfolgerin Shpresa Dafoli im PRZ, sondern auch gleich das Strategie-Muster in die Pkcrypt-Klasse ein. Einige falsche Implementierungen konnte ich auch aus dem C++ Quelltext entfernen, jedoch befinden sich noch immer einige offene Punkte im Programm, die ich noch gerne bereinigt hätte. Eine der neuesten Entwicklungen ist die Klasse “Config”, die für das Einlesen der Konfigurationsdateien sorgt. Damit ist aus dem SME ein Programmsystem geworden, daß man tatsächlich in binärer, also übersetzter Form weitergeben kann und jeweils auf die neuen Gegebenheiten einstellen kann. Das Konfigurationskonzept ist durchaus ausbaufähig. Diverse Verhaltensmuster der Programme könnten darüber noch gesteuert werden. Dazu aber mehr im Kapitel „Aussichten“ auf Seite 135.

Das Ergebnis waren im Grunde zwei völlig neue Programme SMS und SMG. Das “Basteln” an den alten Prototypen wäre sicherlich schneller gegangen und ich hätte auch noch einige Programme mehr aus dem Boden stampfen können. Ich denke jedoch, daß es mir meine Nachfolger im PRZ danken werden, daß ich nicht immer wieder den funktionierenden Prototypen weiter verwendet habe, sondern an entscheidenen Stellen immer wieder neue Programme erstellt habe.

Was soll diese Geschichte zeigen? In vielen Projekten, so wie in diesem, läßt sich nicht vorhersagen, in welche Richtung sich die Software entwickeln wird. An einer gewissen Stelle jedoch zahlt sich die Arbeit wieder aus, die man an einer anderen Stelle für das neue Design aufgewendet hat. Dabei geht es nicht nur darum, bestimmte unschöne Implementierungen zu beseitigen, sondern auch von Zeit zu Zeit das gesamte Konzept zu überdenken und sich auch nicht davor zu scheuen, gravierende Änderungen an der Architektur vorzunehmen. Voraussetzung dafür ist ein geduldiger Projektleiter, der die Vorteile dieses Vorgehens erkennt (immerhin kann man den Projektpartnern oder Auftraggeber in der gesamten Zeit keine neuen Funktionen des Produktes präsentieren) und eine wie auch immer geartete Versionsverwaltung, die es dem Entwickler ermöglicht, zu einem bestimmten Zeitpunkt umzukehren, falls der neue Weg eine Sackgasse war (was man mit einer guten Analyse eigentlich fast ausschließen können sollte) oder falls für andere Module ein noch intaktes Laufzeitsystem benötigt wird. So kann parallel mit dem alten Prototypen weiter gearbeitet werden, während der neue entsteht. Hierbei helfen Entwicklungswerkzeuge, die eine Versionsverwaltung zulassen.

9.1.2 Die Struktur der Quelltexte

Wie bereits angedeutet, gliedere ich meine Quellen in die Bereiche:

- Secure Mail Server (SMS),
- Secure Mail Gateway (SMG),
- Windows Secure Proxy Tools (WINSPT),
- und der SME Bibliothek (SMELIB).

Hinzu kommt noch die sog. Testsuite, die Programme zum Testen der SME Bibliothek enthält. Die Header-Files (.h-Dateien) habe ich in ein gesondertes Verzeichnis untergebracht. Möchte man nur an den Modulen SMS, SMG oder WINSPT arbeiten, genügt eine fertig erstellte Bibliothek und die Header-Files zum Erstellen der Programme.

Aus Gründen der Portierbarkeit tragen alle C++-Quellen bei mir die Endung “.c” und alle Header-Files die Endung “.h”. Ich hatte immer wieder Probleme mit der Endung “.cc”, wenn ich unter Windows die Software übersetzen wollte, und die Endung “.cpp” ist wiederum sehr untypisch für den GNU C++ Compiler unter UNIX. Ich machte also den Kompromiß “.c”.

Jede Klasse ist in einer Datei ihres Namens deklariert und implementiert. Das heißt, daß z.B. die Klasse Mail in der Datei “mail.h” deklariert und in der Datei “mail.c” implementiert ist. Die Dateinamen sind jeweils klein geschrieben. Auch das sind Folgen der DOS-Portierung. Kommentare zur Funktionalität einer Methode stehen in der Regel in den entsprechenden Header-Files der Klasse.

Jede Datei trägt einen Header, der Auskunft über ihre Version, die letzten Änderungen und ihren Namen enthält. Das ist besonders praktisch, wenn diese Dateien irgendwo in gedruckter Form vorliegen.

9.1.3 Weitere Dateien

Mit der Übersetzung des Programms ist es jedoch noch nicht getan. Neben den C++ Quellen benötigt man für das Betreiben eines SME noch einige Procmail-Skripte, Shellscripts und Konfigurationsdateien.

Diese Dateien befinden sich in der Binärdistribution der Bibliothek, da besagte Shellskripte entweder nur Hilfsprogramme für die Installation oder die Wartung des Systems sind und später hoffentlich durch “echte Programme” ersetzt werden, und die anderen Skripte nur zur Konfiguration und zur Kontrolle der eigentlichen Binaries da sind und keine echte Funktion übernehmen.

Ferner existiert ein Verzeichnis “doc/”, in dem ich Kommentare zu Änderungen in den Versionen der Programme untergebracht habe und wo Platz für Kommentare ist. Dieses Verzeichnis dient vor allem dem Verständnis anderer Programmierer. Hier befindet sich keine Beschreibung des Programms oder ähnliches.

9.1.4 Konventionen in den Quelltexten

Wie bereits erwähnt, sollten die Quelltexte auch für europäische Projektpartner zugänglich gemacht werden. So sind die Quelltexte vollständig in Englisch gehalten. Das trifft auch auf die Kommentare im Programm zu. Ein Teil der Diagramme, die Teil dieser Arbeit sind, liegen in älteren Versionen auch in Englisch vor, wurden von mir jedoch nicht weiter gepflegt und sind deshalb nicht Bestandteil der beiliegenden CD.

Zur besseren Lesbarkeit der Quelltexte habe ich mich an folgende Konventionen gehalten:

- Klassennamen fangen mit einem Großbuchstaben an und sind ansonsten klein geschrieben.
- Attribute und Variablen sind vollständig klein geschrieben.
- Über Precompiler-Derektiven erstellte Macros (#define-Anweisungen) sind in Kapitalen geschrieben.
- Methoden und Funktionen beginnen mit einem Kleinbuchstaben. Bestehen diese aus mehreren Wörtern, so fängt das jeweils nächste Wort mit einem Großbuchstaben an.

Die Einrückungen und Formatierungen habe ich versucht, weitgehend nach einem Styleguide der GNU durchzuführen. Bei dieser Art von Formatierung hilft auch der GNU Emacs Editor.

Oft sind die Quelltexte breiter als 80 Zeichen. Das liegt daran, daß ich nur selten Ausdrucke von den Quellen gemacht habe. Meist habe ich die Quelltexte nur auf dem Bildschirm bearbeitet. Für einen Ausdruck empfiehlt es sich, alle Header-Files hochkant und alle C-Files quer zu drucken. In diesem Fall stimmen alle Formatierungen wieder.

9.2 Versionskontrolle

Im Grundstudium hatte ich mehr Interesse daran, möglichst schnell an einem Projekt arbeiten zu können, als nur ein Praktikum zu machen. Ich hatte das Glück, einer der wenigen in einem UNIX-Projekt über ein Jahr sein zu können. Das brachte selbstverständlich das Pech mit sich, daß dieses Jahr ein schwarzes Jahr für andere Veranstaltungen war, aber es brachte mir tiefe Einsichten, die mir später helfen sollten. Eine von diesen Einsichten war die Versionskontrolle.

Zu einem Zeitpunkt, als mein Team die ersten lauffähigen Versionen der geforderten Software entworfen hatte, beging ein Teammitglied einen fatalen Fehler, der sich aber erst zu einem späteren Zeitpunkt zeigte, so daß wir bereits die wenigen älteren Versionen, die wir noch von unseren Quellen hatten, überspielt hatten. Der Fehler war nur mit mühevoller Handarbeit wieder rückgängig zu machen. Eine Katastrophe für das gesamte Projekt.

Das Prinzip von Versionskontrollsystemen ist es, den Stand einer Arbeit zu einem bestimmten Zeitpunkt einzufrieren, um später darauf zurückgreifen zu können. Ferner kontrolliert so ein System in der Regel die Veränderung der Datei von verschiedenen Autoren. Das heißt, wenn Autor A an einer Datei arbeitet, darf Autor B die Datei nur lesen oder Autor A und Autor B dürfen beide daran schreiben, müssen sich danach aber einigen, wie man beide Änderungen zusammenbringen kann. Auch dient so ein System dazu, den Verlauf einer Arbeit zu dokumentieren und im besten Fall sogar wiederzugeben, warum welche Änderungen nötig waren.

Die meisten Systeme dieser Art sind recht teuer. Es gibt jedoch auch zwei mir bekannte Programme, die der GNU General Public Licence unterliegen, also kostenlos erhältlich sind. Das ist zum einen das RCS von Walter F. Tichy und der dafür entwickelte Aufsatz CVS von Dirk Grune, Brian Berliner und Jeff Polk.

Während sich RCS mit der Versionskontrolle von im Grunde einer Datei beschäftigt und diese verwaltet, arbeitet CVS mit Modulen. Jeweils gibt es die Grundfunktionen des sog. Eincheckens und des Auscheckens einer Version. Nur wird dies in der Regel bei RCS für jede Datei gemacht und bei CVS für jedes Modul. RCS ist in der Vergabe der Schreibrechte sehr restriktiv. Es läßt grundsätzlich nur zu, daß ein Autor Schreibrechte auf eine Datei hat. Die anderen Autoren können die Datei lediglich lesen. CVS versucht von Hause aus, die Änderungen von verschiedenen Autoren zusammenzuführen, was funktioniert, solange nicht an der gleichen Stelle im Quelltext editiert wurde. Dann fordert CVS die Hilfe eines Programmierers an, der die Konflikte per Hand beseitigen soll.

Gegen Ende meiner Arbeit am SME mußte ich zwei Tatsachen berücksichtigen:

1. Ich wollte sowohl zu Hause am System arbeiten können als auch im PRZ und

2. meine Nachfolgerin wollte auch so schnell wie möglich eine jeweils aktuelle Version in der Hand halten. Die besten Voraussetzungen hierzu bietet CVS, das in der Lage ist, Module auch über das Internet auszuchecken, sofern entsprechende Zugriffsrechte bestehen. So kann auch über Entfernungen hinweg jedes Teammitglied auf dem Laufenden gehalten werden. Einzige Voraussetzung: Jede lauffähige Version eines Moduls wird wieder dem CVS übergeben, sobald sie getestet ist, und die Teammitglieder sehen regelmäßig nach Neuerungen und vergessen auch sonst nicht, miteinander zu sprechen [Seim97].

Es gibt diverse Erweiterungen für CVS und auch Schnittstellen zum GNU Emacs Editor, der dann bei der Versionsverwaltung hilft. Es gibt graphische Benutzerschnittstellen zu CVS für UNIX, Windows, Apple Macintosh u.v.a.m.

Der gesamte letzte Abschnitt der SME-Entwicklung ist also in einem Repository (so heißt die Datenbank, die die Versionen bei CVS verwaltet) aufgezeichnet. Beim Einchecken von jeder neuen Version in das Repository ist der Benutzer aufgefordert, einen Kommentar zu seinen Änderungen einzugeben. Genau diese Änderungen sind es, die automatisch in den Kommentarzeilen am Kopf jeder Datei mitprotokolliert werden.

Sicherlich wird das Repository in nächster Zeit von meinem Rechner hier zu Hause auf ein System in der TU Berlin verlegt werden (oder es wird dorthin gespiegelt). Dort ist dann der Grundstock für eine Weiterentwicklung gelegt.

9.3 Fehlerverwaltung

Wo Programme geschrieben werden, bleiben leider auch Fehler nicht aus. Gründe dafür gibt es viele. Wichtig ist bei einem größerem Projekt aber nicht nur, Fehler von Anfang an zu vermeiden, sondern auch entdeckte Fehler auf professionelle Weise zu beseitigen.

An erster Stelle steht hier das Aufspüren von Fehlern. Dies kann nach verschiedenen Ansätzen geschehen, die parallel durchgeführt werden können. Der eine, eher konstruktive Ansatz geht davon aus, daß die Programme mit geeigneten Testdaten überprüft werden. Das klingt so weit einleuchtend, schließlich sollte ein Programmierer die Schwächen seines Programms selbst am besten kennen. Leider besteht das größte Problem bei diesem Konzept jedoch darin, geeignete Testwerte zu finden. Außerdem ist es in der Regel so, daß die Probleme nicht die Schwächen darstellen, an die der Programmierer gedacht hat, sondern diejenigen Probleme, die nicht bedacht wurden. Für die einzelnen Module des SME gibt es, wie schon erwähnt, Testprogramme, die die Funktionsweise der Klassen gesondert testen. Außerdem gibt es ein, wenn auch nicht schriftlich fixiertes Testszenario, daß ich beim Testen der Programme immer wieder durchgehe. Dies umfaßt Spezialfälle, wie das Versenden von Mails mit leeren Mailbodies oder die Angabe ungewöhnlicher E-Mailadressen.

Fehler, die jedoch beim Entwurf entstanden sind oder evtl. noch früher, fallen nur beim ausreichenden Testen auf. Es gibt Untersuchungen darüber, wieviel Tester, wieviele Tests durchführen müssen, um möglichst alle Fehler in einer Software zu finden. Für diese Arbeit ist nur wichtig zu wissen, daß Fehler vor allem durch Anwendungen oder durch simulierte Anwendungen gefunden werden können. Bevor das SME also in irgendeiner Verwaltung zum Einsatz kommt, sind diverse Testläufe, möglichst von verschiedenen Personen und auf verschiedenen Systemen nötig, um zunächst alle Fehler zu finden und diese später beheben zu können.

Aber vor der Behebung folgt die Problematik der Erfassung. Nachdem ein Fehler gefunden ist, muß er in einer geeigneten Form erfaßt werden. Auch hierzu gibt es viele Varianten. Die Ideen reichen von Videoaufzeichnungen bis hin zu weitreichenden Dateiaufzeichnungen sämtlicher Mausbewegungen auf dem Bildschirm. Diese Methoden sind für unseren Rahmen alle etwas zu hoch gegriffen. Deshalb führte ich das System GNATS ein.

GNATS ist ein sogenanntes Trouble Ticket System. Ein Benutzer oder Tester füllt, sobald er einen Fehler gefunden hat, ein sogenanntes Trouble Ticket aus. Im Falle des SME gibt es hierfür eine WWW-Seite, auf der die Daten eingegeben werden können. Wichtig dabei ist, daß die Form der Eingabe formalisiert ist. So wird danach gefragt, ob es sich um einen Softwarefehler handelt, einen Dokumentationsfehler oder ob es lediglich ein Wunsch nach Änderungen ist (das kommt vor, wenn das System zwar seinen Zweck erfüllt, sich aber z.B. herausstellt, das es an bestimmten Punkten sehr unergonomisch reagiert). Ferner wird abgefragt, wie schwerwiegend der Fehler ist, ob die Benutzung des Systems durch den Fehler eingeschränkt oder gar unmöglich gemacht wird. Dann wird nach einer Fehlerbeschreibung und einer Beschreibung zur Reproduktion des Fehlers gefragt, und schließlich soll der Benutzer noch sein E-Mailadresse hinterlegen, um über den Stand der Arbeit an diesem Fehler auf dem Laufenden gehalten zu werden.

Ist der Fehler eingegeben, so ist er als Datensatz einer Datenbank angelegt. Er bekommt einen Status (offen), ein Datum und diverse andere Parameter. Der Softwareentwickler bekommt ebenfalls eine E-Mail, die ihm sagt, daß ein neuer Fehler gemeldet wurde. GNATS läßt sich so konfigurieren, daß diese Meldung regelmäßig wiederholt wird, sofern der Entwickler darauf nicht innerhalb einer bestimmten Zeit reagiert. Das macht es im Grunde möglich, das System auch Kunden zugänglich zu machen und eine Antwort des Kundendienstes in einer bestimmten Zeit zu gewährleisten. Im Anhang werde ich eine Liste der zur Zeit bekannten Fehler abdrucken, die durch das GNATS-System erstellt wurde.

Ich halte eine Pflege einer Software nach der Veröffentlichung durch eine solche oder ähnlich geartete Fehlererfassung für unabdingbar.

- Aus der systematischen Erfassung der Fehler können neue Anforderungsdokumente viel leichter erstellt werden als aus ungeordneten Beobachtungen.
- Fehler, die nicht auf das System, sondern auf die Benutzung zurückzuführen sind, können erkannt und z.B. durch bessere Dokumentation oder durch eine ergonomischere Gestaltung der Benutzungsschnittstelle beseitigt werden.
- Die Fehler können kategorisiert werden, so daß aus mehreren Beobachtungen evtl. der gleiche Fehler abgeleitet werden kann.
- Fehler gehen nicht verloren, da explizit zu jeder Fehlerbeschreibung der Status vom Entwickler geändert werden muß.
- Es gibt einen Nachweis darüber, wann welcher Fehler in der Software beseitigt ist, was rechtliche Vorteile bringen kann.
- Auf alte Fehlerbeschreibungen kann zurückgegriffen werden und mit der Beschreibung über die Beseitigung evtl. ein Rückschluß auf aktuelle Fehler gezogen werden.

9.4 Zum Thema Portierung

Eine wichtige Anforderung an das SME war, daß es für Windows und für UNIX benutzbar sein sollte. Dabei sollten die Programme SMG und SMS für UNIX verfügbar sein, aber auch eine Lösung, nämlich das SPT für Windows.

Grundsätzlich gäbe es zum einen die Möglichkeit, nach dem entwickelten Konzept zwei unabhängige Implementierungen für beide Plattformen (UNIX und Windows) durchzuführen. Dabei entstünden aber die Probleme, daß Änderungen, die aus Erfahrungen auf der einen Plattform gemacht wurden, jeweils erst auf der anderen Plattform ebenfalls umgesetzt werden müßten. Man hätte also nicht nur einen initialen doppelten Aufwand, sondern im Grunde ständig den doppelten Programmieraufwand, auch wenn die Konzepte alle wiederverwendet werden können.

Möchte man den Programmtext auf beiden Plattformen verwenden, bleibt keine andere Wahl, als das Programm auf einer Plattform bis zu einem bestimmten Punkt zu erstellen und dann zu portieren.

In [Rud98] wird beschrieben, welche Möglichkeiten bei einer Portierung auf eine andere Plattform existieren:

TABELLE 2. Strategien für Portierung und Multiplattformentwicklung

	Unterstützung einer Plattform	**Unterstützung mehrerer Plattformen**
Portierung existierender Anwendungen	'One-way'-Portierung (in eine Richtung), gefolgt von Einplattformentwicklung	'Coevolutionary' Portierung (in verschiedene Richtungen) gefolgt von Multiplattformentwicklung
Umschreiben existierender Anwendungen	Einplattformentwicklung (Wiederverwendung von Konzept/Design ist möglich)	Multiplattformentwicklung (Wiederverwendung von Konzept/Design ist möglich)
Entwicklung neuer Anwendungen	Einplattformentwicklung	Multiplattformentwicklung

Beim SME habe ich auf eine 'Coevolutionary' Portierung zurückgegriffen, da zunächst die Prototypen für UNIX vorhanden waren und dann die Routinen für Windows erstellt werden sollten. Danach ging die Entwicklung Hand in Hand.

Eine Tatsache, die der o.g. Artikel nicht beschreibt, ist, daß ein Code hinsichtlich seiner Korrektheit durchaus besser werden kann, wenn er auf eine andere Plattform portiert wird. Der Grund ist ganz einfach. Gerade UNIX und Windows unterscheiden sich ganz erheblich von ihrem Systemverhalten. Und so deckt eine Portierung nach Windows beispielsweise Speicherfehler auf, die unter UNIX kaum aufgefallen wären. "Kaum" heißt aber nicht "nie". So können Fehler im Vorfeld vermieden werden, die sonst erst sehr viel später erkannt worden wären. Das heißt jedoch nicht, daß sich nicht durch die Portierung andere Fehler einschleichen können.

Wie im besagten Artikel beschrieben, kann ein Softwareprojekt als Tripel 'Sprachen, Bibliotheken, Werkzeuge' beschrieben werden.

Hätte ich mich für die Verwendung von Java entschieden, hätte ich wohl keine Probleme mit der Portierbarkeit des Codes gehabt. Dafür wären die Probleme bei der native C-Schnittstelle aufgetreten. Ich bin auch heute noch der Meinung, daß eine Portierung der C++-Quellen nach Java auch nachträglich keine größeren Probleme darstellen sollten, da ich auf alle speziellen C-Eigenschaften als Seiteneffekt verzichtet habe und versucht habe, so weit wie möglich Zeiger, Templates o.ä. zu vermeiden, um z.B. genau diesen Weg offen zu halten.

Unser Softwareprojekt beschrieb das Tripel 'C++, Secude/STL, GNU C++'. Auf den ersten Blick sieht das sehr gut aus, denn:

- C++ ist zwar nur bedingt portabel, existiert aber für Windows und UNIX gleichermaßen.
- Secude ist eine Bibliothek, die sowohl für die meisten UNIX-Derivate als auch für Windows zur Verfügung steht. Auch die Standard Template Library (STL) existiert für beide Systeme.
- Auch der GNU C++ Compiler existiert für Windows und UNIX.

So einfach sollte es jedoch nicht werden. Zwar ließen sich die Quellen mit erstaunlich wenigen Änderungen auch auf Windows übersetzen und zum Laufen bringen, es stellte sich jedoch heraus, daß

1. keine Windows-typischen Oberflächen mit dem GNU C++ Compiler zu erstellen waren (vielleicht lag das ja nur an meinem Unvermögen, die Möglichkeiten hierfür aufzutun) und

2. die Secude-Bibliothek nicht für den GNU C++ Compiler verfügbar war, sondern nur für die kommerziellen Compiler unter Windows.

Mit anderen Worten: Ein anderer Compiler unter Windows mußte her. Weil ich selbst stolzer Besitzer eines Symatec C++ Compilers bin und damit schon die Erfahrung gemacht hatte, daß sich dieser sehr UNIX-ähnlich verhielt (zumindest die richtigen Bibliotheken zur Verfügung stellte), machte ich mich an eine Portierung nach Symantec C++.

Das Entsetzen folgte sofort! Die Klasse “String” war dem Compiler nicht bekannt. Ein Blick in die STL-Spezifikation zeigte nun: Die Klasse “String” ist kein fester Bestandteil der STL und somit in keinster Weise standardisiert. Somit gibt es für Symantec C++ keine String Klasse, so wie sie der GNU C++ liefert. Ein Blick in die Handbücher des Visual C++ von Microsoft und des Borland C++ Compilers zeigten mir, daß auch hier keine solchen Klassen vorhanden sind. Auf der Suche nach dem String fand ich dann im Internet eine Klasse, die auf STL aufbaute, für fast alle Compiler unter Windows verfügbar war und zudem fast alle Funktionen, die ich in den Programmen verwendet hatte, nur auf andere Weise implementierte.

Grundsätzlich hat man bei der Portierung einer Software im Falle einer Differenz zwischen den System immer folgende Wahl:

Entweder man greift auf eine Multiplattformbibliothek, wie z.B. STL zurück, dann muß man dies aber auch gründlich tun und nicht so wie ich, der dabei übersehen hat, daß er letzten Endes etwas zu viel von der Bibliothek erwartet, oder man schreibt selbst eine Multiplattformbibliothek. Das habe ich im Grunde mit der SME Bibliothek getan. Die Klassen dieser Bibliothek sind jetzt für beide Plattformen zugänglich. Die andere Möglichkeit ist eine sog. Mapping-Bibliothek, also eine Bibliothek, die sich genau so verhält, wie unter der bereits implementierten Plattform, nur mit dem Unterschied, daß sie in Wirklichkeit andere Funktionen aufruft, die wiederum auf dieser Plattform existiert.

Ich entwickelte also eine Mapping-Klasse “String”. Für die SME Bibliothek sieht diese Klasse aus, wie die von UNIX bekannten Strings. In der Realität rufen die Methoden jedoch wieder Methoden der bstring-Klasse auf. Das ist die Klasse, die auf STL aufsetzt und für diverse Windows C++-Compiler verfügbar ist.

Die Kunst bestand nun darin, nicht nur syntaktisch das gleiche Bild der Klasse nach außen zu erzeugen, sondern auch die gleichen Ergebnisse zu liefern, wie die UNIX-Klasse. Aber damit nicht genug. Es stellte sich ferner heraus, daß diverse Funktionen, die auf Zeichenketten (char-Arrays) unter UNIX definiert sind, in den Routinen des Windows C++-Compilers nicht existierten.

Bei der Entwicklung der Windows-Oberfläche unter Symantec C++ tauchten plötzlich Probleme auf, die bei einer Anfrage bei Symantec das Ergebnis erbrachten, daß dieser Compiler nicht länger gepflegt wird.

Ich befand mich also in einer Sackgasse. Zwar hätten sich die Probleme umgehen lassen können, aber die Entwicklung des SME sollte irgendwann weiter gehen und ich konnte das System nicht in einem Dialekt stehen lassen, für den es gar keinen Compiler mehr gibt.

Also beschaffte ich mir einen Microsoft Visual C++ Compiler und fing mit der Portierung von Symantec C++ nach Visual C++ an. Wer meint, daß dies nicht so schlimm gewesen sein kann, da es sich dabei um Compiler auf der gleichen Plattform handelt, dem sei geraten, es selbst einmal auszuprobieren.

Die SPTs bestehen aus einem Proxy für das Senden von Nachrichten und einem für das Empfangen. Damit nicht das Starten dieser beiden Programme ein neues Anzeigen der Secude Copyrightmeldung zur Folge hat, mußten beide Funktionen in ein Programm. Von UNIX her bin ich das gewohnt, durch die Funktion fork() zu lösen. Das Programm spaltet sich von diesem Moment an in zwei Prozesse und jeder Prozeß kann seinen Aufgaben nachgehen. Die Funktion fork() existiert aber unter Visual C++ nicht. Stattdessen muß hier entweder auf das Task- oder das Thread-Konzept von Windows zurückgegriffen werden.

Die parallele Benutzbarkeit der SPTs unter UNIX und Windows hatte sich damit völlig zerschlagen. Was jetzt noch möglich wäre, wäre eine Isolierung der eigentlichen Proxy-Funktionen vom Rest des Programms. Solange das Programm aber so monolithisch bleibt, wie ich es im Moment entworfen habe, gibt es keine Möglichkeit, eine Multiplattformentwicklung des SPT durchzuführen.

Das ist aber nicht weiter tragisch, da unter UNIX in jedem Fall die SMG-Software mit fast der gleichen Funktionalität eingesetzt werden kann. Auf der anderen Seite kann das SMG und das SMS auch unter Windows zum Laufen gebracht werden. Voraussetzung hierfür ist nur, daß eine Konfiguration eines SMTP-Servers und POP3-Servers möglich ist, die Filterprogramme wie unter UNIX zulassen. Da es eine Portierung zumindest von Sendmail nach Windows NT gibt, bin ich zuversichtlich, daß auch einer solche Anwendung der Serverkomponenten unter Windows NT zumindest nichts mehr im Wege steht.

Der Stand ist also: Wir haben eine gemeinsame Codebasis bestehend aus SMELIB, SMS, SMG, INCLUDE und eine Einplattformentwicklung von WINSPT mit der Option, auf dieser Basis auch beispielsweise ein XWINSPT erstellen zu können.

9.4.1 Weitere Compilerfallen

Bei der Portierung nach Visual C++ sind noch weitere Probleme aufgetreten, die ich im folgenden beschreiben möchte:

Visual C++ hat eine andere Strategie bei der Auswahl der überladenen Methoden. Das sorgte bei der Portierung dafür, daß automatische Umwandlungen nicht funktionierten oder Aufrufe von überladenen Methoden praktisch im Kreis durchgeführt wurden. Wie nicht anders üblich, hatte ich die Methode mit einem Typen implementiert und die anderen Methoden eine Typenwandlung durchführen lassen, die dann wiederum die implementierte Methode aufriefen. Durch eine falsche Auswahl im Visual C++ wurde jedoch nicht diese sondern eine ebenfalls nur aufrufende Methode ausgewählt usw. Die nun in großen Mengen im Quelltext vorhandenen Casting-Operatoren, die keine andere Auswahl mehr zulassen, tragen nicht gerade zur besseren Lesbarkeit bei.

Ein kleines Beispiel:

Der ursprüngliche Quelltext beinhaltete eine Reihe von Zeilen der Form

```
if (downcase(name) == "to:") {
  to = body;
  return;
}
```

Der GNU C++ Compiler macht hier einige automatische Typenwandlungen. Die Funktion downcase() ist auf dem Datentyp char* definiert, ebenso könnte “to:” als char* aufgefaßt werden. Leider existiert kein Operator ==, der als (char*) == (char*) definiert ist, wohl aber ein (String) == (String). Ferner existiert eine Typenwandlung von char* nach String. Somit übersetzte der GNU C++ Compiler mit dem (String) == (String) Operator und machte eine automatische Typenwandlung nach String.

Die unter Visual C++ laufende Variante sieht so aus:

```
    String tmpname(name);
    tmpname.downcase();

    if (strcmp(tmpname, "to:") == 0) {
      to = body;
      return;
    }
```

Ich benutze hier die alte C-Funktion strcmp(), die auf den Datentypen char* definiert ist. Da die Variable name aber nicht innerhalb von strcmp() automatisch in char* umgewandelt wird (das liegt vielleicht daran, daß name nicht als String sondern als const String definiert ist), mache ich zuerst eine Variablenzuweisung nach tmpname, wandle dann tmpname in Kleinbuchstaben um und übergebe tmpname als Parameter an strcmp(). In diesem Fall greift dann die automatische Typenumwandlung.

Weitere Probleme tauchen immer wieder bei der Speicherverwaltung auf:

Visual C++ 5.0 akzeptiert zwar Zeichenfelder beliebiger Größe, kann sie aber offensichtlich nicht verwalten. Hier ein kleines Programm, daß den Effekt demonstriert:

```
#include <iostream.h>
#include <malloc.h>

void strangeFun()
{
  char *c = (char *) malloc(2000000);
  cout << "Diese Funktion ist komisch." << endl;
}

void reallyStrangeFun()
{
  char c[2000000];
  cout << "Diese Funktion auch." << endl;
}

void main()
{
  cout << "Demo für Arrayverhalten" << endl;
  strangeFun();
  reallyStrangeFun();
  cout << "Ende." << endl;
}
```

Die Funktion strangeFun() wird ausgeführt. Die Funktion reallyStrangeFun() stürzt ab. Das ist völlig unabhängig von der Reihenfolge oder vom Kontext.

Die Socketfunktionen, die von UNIX her bekannt sind, gibt es zwar unter Windows auch, nur muß hier vor der Benutzung von Sockets die entsprechende DLL (Dynamical Link Labrary) initialisiert werden.

Der folgende C++ Quelltext führte unter Windows zum Absturz:

```
ifstream ifs(filename);
String s, t;
char *desc;
char c;

if (ifs == NULL)
    return;

while (!ifs.eof()) {
  s="";
  t="";

  while ((ifs.get(c)) && (c != '=')) {
    if (c >= ' ') s += c;
  }
  if (s != "") {
    while ((ifs.get(c)) && (c != '\n')) {
          if (c >= ' ') t += c;
    }
    if (t != "") {
         desc = mallocString((const long) s.length());
         strcpy(desc, s.chars());
         cm[desc] = mallocString((const long)
                                 t.length());
         strcpy(cm[desc], t.chars());
    }
  }
}
ifs.close();
```

Unter UNIX läuft diese Funktion einwandfrei. Unter Windows läuft sie nur, wenn man den Quellcode für DOS übersetzen läßt und nicht als Win32-Applikation, was aber für die Erstellung einer Fensteroberfläche nötig ist. Ich ersetzte den Code, durch sein C-Pendant:

```
FILE *fp;
String s, t;
char *desc;
int c;
```

```
if ((fp = fopen(filename, "r")) == NULL)
  return;
  while (feof(fp) == 0) {
    s="";
    t="";
    while ((c = fgetc(fp)) && (c != '=') && (feof(fp) == 0)) {
      if (c >= ' ') s += c;
    }
    if (s != "") {
      while ((c = fgetc(fp)) && (c != '\n') && (feof(fp) == 0)) {
        if (c >= ' ') t += c;
      }
      if (t != "") {
        desc = mallocString((const long) s.length());
        strcpy(desc, s.chars());
        cm[desc] = mallocString((const long) t.length());
        strcpy(cm[desc], t.chars());
      }
    }
  }
fclose(fp);
```

Eine niemals endende Geschichte bei der Portierung zwischen DOS und UNIX werden die Zeilenenden sein. Während UNIX seine Zeilen mit einem sog. “Newline”-Zeichen (ASCII-Wert 10) beendet, enden die DOS-Zeilen auf “Carriage Return” und “Newline” (ASCII-Wert 13 und 10). Bei der Überprüfung der Signaturen muß aber eine absolute Übereinstimmung also auch inklusive Zeilenenden bestehen. Ein nicht ganz unwesentlicher Teil in einigen Funktionen beschäftigt sich daher damit, “Return”-Zeichen zu filtern oder hinzuzufügen.

Richtig ärgerlich wird die Situation aber erst dann, wenn man versehentlich ein Makefile (ein solches enthält Vorschriften für die Erstellung eines komplexen Compilats inklusive Abhängigkeiten der Dateien untereinander) für UNIX mit einem DOS-Editor geöffnet hatte. Das Programm make(1) liefert daraufhin völlig unverständliche Fehlermeldungen. Aus irgendeinem mir nicht verständlichen Grund zeigen der vi(1) und der xemacs(1) Editor unter Linux (Distribution S.u.S.E. 5.0) keine Kontrollzeichen an. Damit war es mir nicht möglich, die “Return”-Zeichen im Makefile überhaupt zu sehen. Dementsprechend lange dauerte die Suche des Fehlers. Bei der Ausgabe der Fehlermeldungen von make(1) entstand nur deshalb eine völlige Verwirrung, da das Make-Programm versuchte, die fehlerhafte Zeile mit auszugeben. Dabei enthielt die Meldung jedoch das Steuerzeichen “Return”, das dazu führte, daß die Schreibmarke wieder am Anfang der Zeile stand und somit den Anfang der Zeile überschrieb.

Abhilfe bei solchen Problemen schafft ein Programm der S.u.S.E. Linux-Distribution mit der Bezeichnung “dos2unix”. Dieses dient als Filter. So lassen sich Dateien mittels “dos2unix < dosfile > unixfile” von lästigen Steuerzeichen befreien. Ferner kann das Programm auch Umlautkonvertierungen durchführen (DOS verwendet eine eigens entwickelte Zeichentabelle oberhalb des ASCII-Wertes 127, UNIX verwendet den ISO-Latin1-Standard).

KAPITEL 10

Installation und Konfiguration

Man sollte meinen, daß an dieser Stelle über die erstellte Software genug gesagt ist. In diesem Fall ist das jedoch nicht so. Die Installationsprozedur und die Konfiguration der einzelnen Komponenten sind ein wichtiger Bestandteil dieser Arbeit und dürfen auf keinen Fall unterschätzt werden. Die erstellten Software-Module können nur bei einwandfreier Konfiguration die gewünschte Sicherheit erbringen. Fehler äußern sich evtl. nicht durch Fehlermeldungen, Abstürzen oder Zeitüberschreitungen, sondern im schlimmsten Fall nur dadurch, daß eine Nachricht ungesichert oder nicht ausreichend gesichert übertragen wird. Aus diesem Grund muß an dieser Stelle die Installation und Konfiguration genauestens betrachtet werden.

Eine Anforderung an das System war ein möglichst geringer Administrationsaufwand. Im Punkt Installation ist dieses Ziel nicht ganz erreicht. Ein Grund hierfür ist ein Konflikt zwischen der Anforderung “Sicherheit” und “leichte Wartbarkeit”. Würde man ein Installationsprogramm entwickeln, das wirklich alle Arbeit bei der Installation erledigt, bestünde die Gefahr, daß ein Fehler hierbei später fatale Folgen hätte. Das Problem ist vor allem die Umgebung, in die ein SME eingebettet wird. Könnten wir von einer einzigen Systemarchitektur mit einer genau definierten Installation ausgehen, ließe sich auch die Installation einfach gestalten. Da die Vorgabe aber “UNIX-System” heißt, müssen wir alle Detailfragen an den Systemadministrator übergeben, der von der Qualifikation her dazu in der Lage sein müßte, das System korrekt aufzubauen. Ferner können wir dem Administrator Hilfsmittel geben, die ihm bei der Konfiguration helfen.

Ich selbst arbeite seit meiner Schulzeit mit UNIX-Derivaten und kenne daher die Anforderungen, die verschiedene Systeme an den Administrator stellen. Gemessen an den ersten Versionen von CNEWS auf MINIX oder z.B. der Integration eines neuen PCMCIA-Controlers unter LINUX ist die Installation eines SMS ein Kinderspiel. Wichtig dabei ist vor allem, die Prinzipien zu verstehen, die dahinter stekken.

10.1 Grundlagen

Das SME ist in eine Reihe von anderen UNIX-Werkzeugen eingebettet. Wie beschrieben, greife ich auf Funktionen des Gespanns Sendmail / Procmail zurück. Ferner benötigen wir zum Betrieb das Secude-Paket für die Verschlüsselungsfunktionen. Vor allem als SMTP-Server kommen auch andere Pakete, wie z.B. SMail in Frage. Ich möchte mich hier jedoch bei meiner Beschreibung auf die eine, getestete Variante beschränken.

10.1.1 Sendmail

Bei der von mir beschriebenen Variante handelt es sich um Sendmail 8.8.5. Abweichende Versionen dürften hier keine Schwierigkeiten bereiten, solange sie die “Features” unterstützen, die für das SME benötigt werden. Ältere Versionen von Sendmail machten es nötig, die Datei sendmail.cf anzupassen. Diese Datei ist aber selbst für hartgesottene UNIX-Gurus schwer handhabbar. Deshalb wurde die Verwaltung über einen Macro-Compiler eingeführt (M4-Compiler) [Cost97]. Eine Sendmail Konfiguration kann z.B. folgendes Aussehen besitzen:

```
include(`../m4/cf.m4')
VERSIONID(`linux for smtp-only setup')dnl
OSTYPE(linux)dnl
define(`confDEF_USER_ID', `daemon:daemon')dnl
define(`PROCMAIL_MAILER_PATH', `/usr/bin/procmail')dnl
define(`QUEUE_DIR', `/var/mqueue')dnl
define(`confTRUSTED_USERS', `uucp mdom wwwrun')dnl
FEATURE(local_procmail)dnl
FEATURE(nouucp)dnl
FEATURE(always_add_domain)dnl
MAILER(local)dnl
MAILER(procmail)dnl
MAILER(smtp)dnl
```

Die erste Zeile ist im Grunde nur für den M4-Compiler interessant. Die zweite Zeile beschreibt die Konfiguration und taucht so auch wieder in der generierten sendmail.cf-Datei auf, so daß es möglich ist, vom Compilat auf die Quelle zu schließen. VERSIONID ist also für den Administrator bestimmt, um Konfigurationen wiederzufinden und zu identifizieren. OSTYPE gibt die Plattform an, für die das Macro geschrieben ist. In diesem Fall “linux”.

Es folgen einige Definitionen. Hier sind z.B. Benutzerkennungen und Dateipfade angegeben, die für Sendmail von Bedeutung sind.

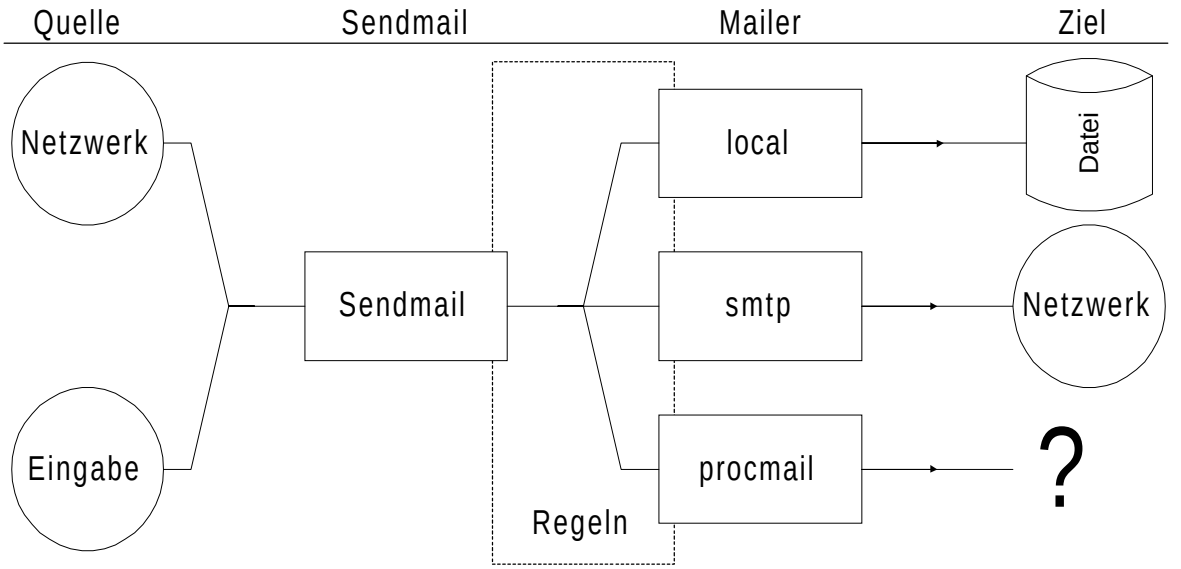
Sendmail verfolgt ein Konzept, bei dem sog. Features ein oder ausgeschaltet werden können. Diese werden durch Zeilen beginnend mit FEATURE angegeben. Hier sollte das Feature “local_procmail” enthalten sein. Dies zieht mit sich, daß die Definition “PROCMAIL_MAILER_PATH” ebenfalls gesetzt ist. Im Grunde dient diese Konfiguration nur dazu, daß beim Ausliefern auf der lokalen Maschine “.procmail”-Dateien der Benutzer ausgewertet werden. Das bietet diverse interessante Möglichkeiten beim Umgang mit E-Mails. Für das Funktionieren des SME benötigt man neben den Mailern “local” und “smtp” zwingend nur die Zeile MAILER(procmail)dnl.

Sendmail versteht sich selbst als Schaltzentrale für die E-Mailauslieferung. Sendmail empfängt Mails und kann diese bestimmten Regeln entsprechend an interne oder externe Module weiterleiten, die für Sendmail die Auslieferung vornehmen. Dabei ist der Mailer “local” für die Auslieferung der Mails auf der Festplatte zuständig und der Mailer “smtp” für die Auslieferung über das Netzwerk.

Es wäre durchaus möglich, die Programme SMS oder SMG bzw. SMG-Send an dieser Stelle als Mailer zu definieren und einen MAILER(sms)dnl einzutragen. Dies hätte den Vorteil, daß Sendmail direkt die Programme starten würde, was performanter und sicherer wäre. Ich verwende jedoch Procmail und ein aus dem Procmail gestartetes Shellscript, um die Vorgänge etwas transparenter zu machen und habe deshalb auf die Procmail-Methode zurückgegriffen.

Wichtig ist: Das M4-Makro aktiviert den Mailer “procmail”. Es muß an einer anderen Stelle jedoch noch konfiguriert werden, wann dieser Mailer benutzt wird.

ABBILDUNG 59. Das Mailer-Modell in Sendmail



Trifft eine Regel zu, übergibt Sendmail die E-Mail über den Standardeingabestrom an das spezifizierte Programm. Außerdem wird als Parameter übergeben, wer der Absender und wer der Empfänger der E-Mail ist. Dies ist vor allem wichtig, wenn die E-Mail an mehrere Personen adressiert ist. In diesem Fall startet Sendmail für jeden Adressaten den den Regeln entsprechenden Mailer. Sendmail erwartet keine Ausgaben oder Rückgabewerte von dem gestarteten Programm [Cost97].

10.1.2 DNS-Konfiguration

Der Nachteil von sehr modular aufgebauten Softwarekomponenten ist die Problematik ihrer Abhängigkeit. Wie beschrieben benutze ich Sendmail zum Empfang und zur Weiterleitung von E-Mails. Sendmail wiederum benutzt den Domain Name Service-Dienst, um diejenigen Rechner ausfindig zu machen, an die eine E-Mail per SMTP ausgeliefert werden muß.

Ich führe diese Tatsache aus zwei Gründen an: Erstens muß eine DNS-Konfiguration existieren, die es dem SMS zuläßt, die Mails über die SMGs auszuliefern, zum anderen müssen alle Mails des SME den SMS erreichen. Zweitens ist es notwendig, die Firewall so zu konfigurieren, daß sie DNS-Anfragen zuläßt. Alternativ kann ein SMG auch so konfiguriert werden, daß er beispielsweise einen eigenen DNS-Dienst anbietet, den er selbst und die hinter dem SMG befindlichen Rechner nutzen können und der zusätzlich nur die IP-Adresse des SMS kennt [Holz98].

Die Arbeitsrechner im gesicherten LAN brauchen nicht zwingend einen Nameserver. Man könnte in den Mailclient-Programmen auch die IP-Nummer des SMG angeben. Damit könnten die Mails bis zum SMG ausgeliefert werden, dieser kann dann die Auslieferung zum SMS durchführen. Dies ist möglich, indem man den SMS ohne DNS konfiguriert (FEATURE(nodns)dnl) und in der Mailertable (siehe unten) eine IP-Nummer direkt in eckigen Klammern statt einem Rechnernamen angibt.

Dann ist es jedoch leider nicht mehr möglich, einen redundanten SMS zu konfigurieren, was sonst möglich wäre.

DNS läßt zu, mehrere redundante SMTP-Server für eine Domäne anzugeben. Dabei wird eine Gewichtung angegeben: Je kleiner der Wert, desto höher die Priorität. Das heißt, daß ein sendender

SMTP-Server versuchen wird, an den SMTP-Server zu senden, der die höchste Priorität hat. Ist dieser aus irgendwelchen Gründen nicht erreichbar, wird an den nächsten SMTP-Server gesendet usw.

Jetzt könnte die Idee entstehen, daß dieses Vorgehen beim SMS sowieso nicht funktionieren würde, weil sich eine Verwaltung nicht mehrere SMS-Server leisten können wird oder will. Das ist auch nicht nötig. Es ist möglich, sendmail so zu konfigurieren, daß es Mails annimmt, um diese bei der nächsten Gelegenheit an einen anderen Rechner weiterzuleiten. Man spricht hierbei von einem sog. Relay-Vorgang. Dabei wird die E-Mail von diesem SMTP-Server gar nicht weiter interpretiert, sondern nur aufgehoben, bis der wirkliche Empfänger-Rechner wieder verfügbar ist. Für den Fall, daß also der SMS irgendwann ausfällt, können andere SMTP-Server benannt werden, die keine SME-Softwarekomponenten benutzen müssen, sondern die E-Mail so lange zwischenspeichern, bis der SMS wieder verfügbar ist [Cost97].

10.1.3 Die Funktionsweise von Procmail

Procmail ist ein Programm zum automatisierten Bearbeiten von E-Mails. Es ist vor allem dazu gedacht, E-Mails automatisch zu sortieren oder zu verändern. So wird Procmail gerne für die Implementierung von sog. Gateways genutzt, wo E-Mails von einem Format in ein anderes gewandelt werden.

Das Konzept, das Procmail dabei verfolgt, ist eine Art Fortsetzung der Konzepte, die in der sendmail.cf existieren. Wie Sendmail arbeitet auch Procmail nach Mustern und Zuständen. Ein Procmail-Script startet in einem Zustand und kann durch Musterregeln (engl. pattern-matching) in einen anderen Zustand gebracht werden und dabei Änderungen an der Eingabe vornehmen. Es ist also so etwas wie eine spezielle Implementierung eines endlichen, deterministischen Automaten. Ein Procmail-Script kann z.B. so aussehen:

```
:0
* ^Subject: *E2S*
$HOME/Mail/e2s

:0
* ^To: *cryptlib@*
$HOME/Mail/cryptlib
```

Beide Regeln starten vom Zustand “0” (gekennzeichnet durch “:0”). Die erste Regel besagt: Wenn die E-Mail eine Zeile beinhaltet, die mit “Subject:” beginnt (der Zeilenanfang wird durch das “^” repräsentiert) und in dieser Zeile die Zeichenkette “E2S” vorkommt, wird sie in das Verzeichnis $HOME/Mail/e2s einsortiert. Die andere Regel besagt, wenn in einer Zeile, die mit “To:” anfängt, die Zeichenkette “cryptlib@” vorkommt, wird sie in das Verzeichnis $HOME/Mail/cryptlib verschoben. Im SME benötigen wir Procmail nur zum Starten von entsprechenden Shellscripts.

10.2 Technische Voraussetzungen

Im folgenden Abschnitt möchte ich eine Aufstellung der technischen Voraussetzungen machen, die man benötigt, um die Systemkomponenten zu betreiben. Dabei greife ich auf einige Erfahrungswerte und begründete Abschätzungen zurück. Um sicher zu gehen, gebe ich zusätzlich auch immer noch ein bis zwei Referenzsysteme an, auf denen ich die Software bereits getestet habe. Bei den Angaben der UNIX-Workstations bleibt es dem erfahrenen Systemadministrator überlassen, eine ähnliche Konfiguration für seine Hardwareplattform auszuwählen. Grundsätzlich gehe ich jedoch davon aus, daß die von mir erstellten Softwarekomponenten eine so geringe Anforderung an die Hardwareplattform stellen, daß praktisch jedes heute gekaufte System in der Lage ist, die E-Mails in ausreichender Geschwindigkeit auch bei höherem Aufkommen zu bewältigen.

10.2.1 Voraussetzungen für den SMS-Server

Als Kernkomponente sind hier die Anforderungen ganz besonders hoch. Über den Einsatz auf einer CMW-Maschine kann ich leider keine Aussagen treffen. Ich habe bislang nur an einer CMW-Maschine gesessen und außer einigen kleinen Tests noch nichts bezüglich dieser Technik gesehen. Die Aussage unserer Projektpartner war jedoch immer, daß die Leistung absolut vergleichbar mit einer unter HPUX 10.x laufenden HP 9000er Maschine wäre. Die Hardware ist die gleiche. Das CMW-System beschränkt die Leistung nicht erheblich.

Auf folgenden Referenzrechnern ist das SMS bereits gelaufen und hat dort bis zu 15 E-Mails pro Minute übertragen (dieser Wert gibt nicht die Maximal-Last an, sondern entspricht nur dem getestetem Aufkommen):

Sparc-Clone unter Solaris

- Twinhead Sparc 10
- 64 MB RAM
- Solaris 2
- 1 GB SCSI Festplatte

Linux-Minimalkonfiguration

- 486DX4-100
- 16 MB RAM
- Linux 2.0.x
- 1 GB SCSI Festplatte

Entwickler-Plattform

- Pentium 90
- 32 MB RAM
- Linux 2.0.x
- 1 GB EIDE Festplatte

Folgende Konfiguration empfehle ich für den Einsatz mit bis zu 30 E-Mails pro Minute Aufkommen (zu Stoßzeiten mehr möglich):

- Pentium 200
- 64 MB RAM
- Linux 2.0.x
- 1 GB UW-SCSI Festplatte

Auf einem solchen System sind bereits Tests gelaufen, jedoch hatte ich noch keine Gelegenheit, ein dauerhaftes E-Mailaufkommen dieser Größenordnung zu testen. Ein Engpaß wird an dieser Stelle schnell die Größe der Festplatte, da die E-Mails alle irgendwo gelagert werden müssen (für Tests habe ich sie nach /dev/null geleitet), erreichen. Die Kryptoalgorithmen stellen für einen Pentium 200 keine nennenswerten Probleme mehr dar. Als Flaschenhals entpuppt sich hier die Festplatte und der SCSI-Controller. Hier sollte bei einem professionellen Betrieb nicht gespart werden!

10.2.2 Voraussetzungen für einen SMG-Server

Da SMGs in größerer Zahl benötigt werden, ist es interessant, wie preiswert ein solches System angeschafft werden kann. Dafür ist hier das Mailaufkommen nicht so von Bedeutung. Da aus beschriebenen Gründen die Zahl der Mitarbeiterinnen und Mitarbeiter hinter dem SMG möglichst klein sein sollte, gehe ich hier von einem Aufkommen von höchstens 6 E-Mails pro Minute aus. Dieses Aufkommen bewältigt die folgende Konfiguration nach meinen Erfahrungen sicherlich (bislang konnte dies nicht getestet werden):

- 486DX2-66
- 16 MB RAM
- Linux 2.0.x
- 1 GB EIDE Festplatte

Als Referenzrechner, auf dem ein SMG schon getestet wurde, gebe ich hier das folgende System an:

- Pentium 90
- 32 MB RAM
- Linux 2.0.x
- 1 GB EIDE Festplatte

Wichtig dabei ist, daß man mitbedenkt, welche anderen Dienste evtl. über die Firewall geleitet werden und wie schnell die Anbindung ist. Die Firewallaktivitäten sind bei schnellen Netzwerkleitungen nicht zu unterschätzen!

10.2.3 Voraussetzungen für einen SPT-Client

Leider sind meine Erfahrungen mit Windows-Rechnern nicht so weitreichend, wie bei UNIX Workstations oder Linux- bzw. FreeBSD-Rechnern. Aus diesem Grund hier nur einige Referenzrechner, auf denen ich die Software getestet habe. Dabei bestimmt die Rechenleistung im Grunde nur, wieviele Mails man pro Minute absenden kann.

- Pentium 133
- 64 MB RAM
- Windows 95
- 1 GB EIDE Festplatte
- ca. 2-3 Mails pro Minute

- Pentium 166 MMX
- 64 MB RAM
- Windows 98
- 3 GB EIDE Festplatte
- ca. 4-5 E-Mails pro Minute

Unter Windows NT erhöhen sich die Anforderungen, da das System hier nach meinen Beobachtungen immer wieder mit dem Auslagern von Speicherblöcken beschäftigt ist. Mit ausreichender RAM-Größe sollten hier sehr viel höhere Geschwindigkeiten erreichbar sein.

- Pentium II 233
- 128 MB RAM
- Windows NT
- 4 GB UW-SCSI Festplatte
- ca. 3-4 E-Mails pro Minute

10.2.4 Voraussetzungen für Rechner hinter einem SMG

An Rechner, die sich hinter einem SMG befinden, werden keine besonderen Anforderungen gestellt. Es kann sich hierbei um beliebige Systeme mit beliebigen Programmen handeln, die SMTP und POP3 unterstützen.

10.3 Installation des SMS

Die Installation des SMS gliedert sich in drei Schritte: Die Konfiguration von Sendmail, die Firewallkonfiguration (falls vorgesehen) und die Erstellung der SMS-Schlüssel zur Verteilung.

Ich gehe davon aus, daß Secude auf dem Rechner installiert wurde, was sich im Grunde darauf beschränkt, das TAR-Archiv auszupacken und ggf. einen Link auf das Secude-Programm zu machen.

Zunächst muß jedoch die SMS Verzeichnisstruktur aufgebaut werden. Eine abweichende Installation ist auch möglich, wird hier jedoch nicht beschrieben:

Unter /usr/local/sms werden die Unterverzeichnisse etc, bin und log eingerichtet. Dann werden die Programme und Skripte aus der Distribution an die entsprechenden Stellen kopiert und mit den korrekten Permissions versehen:

ABBILDUNG 60. Verzeichnisinhalte /usr/local/sms/bin

```
/usr/local/sms/bin:
total 102
-rwxr-xr-x   1 root     root          139 Nov  8 20:46 addpk
-rwxr-xr-x   1 root     root           94 Nov  8 20:45 showpk
-rwxr-xr-x   1 root     root          216 Nov  6 11:47 sms
```

Unter /usr/local/sms/etc wird das Script “smsfilter” untergebracht.

Es bietet sich an, ein Verzeichnis “public-keys” anzulegen, in dem man die öffentlichen Schlüssel der SMGs und SPTs sammelt. Auch der öffentliche Schlüssel des SMS kann darin enthalten sein.

Bei abweichenden Konfigurationen müssen die Skripte “smsfilter” und “sms” angepaßt werden. Ferner muß eine Datei “/usr/local/sms/etc/sms.conf” angelegt werden, die (in angepaßter Form) folgendes beinhalten muß:

ABBILDUNG 61. Die Datei smsrc

```
myname=C=DE,O=TUB,CN=SMS,SN=SYS00
mypin=1639178
psepath=/usr/local/sms/pse
mexuid=sms
sigHdrFailTxt=Beim Ueberpruefen des Mailheaders ist ein Fehler aufgetreten.
sigBdyFailTxt=Beim Ueberpruefen des Mailtextes ist ein Fehler aufgetreten.
secInfoTxt=Die Mail ist sicher und geprueft.
secSubject=Sicherheitsfehler
acmihost=hoogla
```

- **myname** ist der Schlüssel-Identifizierer für den Schlüssel des SMS.
- **mypin** ist die PIN, um den privaten Schlüssel öffnen zu können.
- **psepath** bezeichnet den Pfad zum sog. PSE, der die privaten und öffentlichen Schlüssel enthält.
- **mexuid** Der Name, unter dem sich das SMS beim ACMI anmeldet.
- **sigHdrFailTxt** Der Text, der im Falle eines Fehlers bei der Überprüfung der Header-Signatur zurückgesendet wird.
- **sigBdyFailTxt** Der Text für fehlerhafte Signaturen unter Mailbodies.
- **secInfoTxt** Der Text, der der Mail vorangestellt wird, wenn die Mail korrekt war (kann auch leer sein).
- **secSubject** Die Betreff-Zeile, die eingefügt werden soll, wenn ein Fehlerreport an den Absender zurückgesendet wird.
- **acmihost** Rechnername oder IP-Nummer des Rechners, auf dem das ACMI läuft.

10.3.1 Sendmail-Konfiguration

Die Sendmailkonfiguration beschränkt sich darauf, ein sendmail.cf mit den entsprechenden Features zu erstellen und Mailertable-Einträge der folgenden Form zu machen.

ABBILDUNG 62. Datei /etc/mailertable beim SMS

```
tcmail.tu-berlin.de      local:[127.0.0.1]
sms.tu-berlin.de         procmail:/usr/local/sms/etc/smsfilter
```

Die Konfiguration geht davon aus, daß der Rechner, auf dem das SMS-System läuft, als tcmail.tu-berlin.de bekannt ist. Als solchen sollte sich der SMS selbst kennen. Er trägt aber auch die im Nameserver eingetragene Bezeichnung sms.tu-berlin.de. Je nachdem, unter welchem Namen nun eine Mailauslieferung stattfinden soll, speichert Sendmail die Nachricht auf der Festplatte (local) oder das Procmailprogramm wird gestartet (procmail).

Das SME-Konzept basiert darauf, daß die E-Mails jeweils umadressiert werden, um die richtigen Filter aufzurufen. Genaueres hierzu im Abschnitt „Von Adressen und Mailern“ auf Seite 118.

10.3.2 Firewall-Konfiguration

Eine Firewallkonfiguration ist nur dann nötig, wenn der SMS nicht zusätzlich durch eine Firewall geschützt ist, der Rechner also ungeschützt im Internet steht oder wenn z.B. eine Konfiguration, wie in „Trennung von MH und KH“ auf Seite 34 beschrieben, erstellt werden soll. Pauschal gilt:

- Alle Dienste außer Mail (Port 25), POP3 (Port 110) und DNS (Port 53) sollten gesperrt werden.
- Existiert ein Keyserver auf einer bestimmten Netzwerkschnittstelle, so muß auf dieser auch der Port des Keyservers vorgeschaltet werden. Jedoch muß der SMS-Rechner Verbindungen auf diesem Port nicht entgegennehmen, sondern nur erstellen können.

10.3.3 Schlüsselverwaltung

Mittels Secude muß ein Schlüssel für das SMS erstellt werden. Dies erreicht man durch den Aufruf von Secude

```
% /usr/local/secude/bin/secude
```

Es erscheint:

```
SECUDE 5.0c (c) GMD 1997
http://www.darmstadt.gmd.de/secude
secude-support@darmstadt.gmd.de
GMD grants the right to use this
version of SECUDE solely for
non-commercial purposes.
Type "help" to get a list of available commands.

secude>
```

Mittels des Kommandos:

```
secude> psecrt -p /usr/local/sms/pse
```

kann eine sog. PSE im Verzeichnis von SMS erstellt werden. Secude fragt dann zweimal nach einer PIN. Nach dieser Eingabe muß der Schlüssel-Identifizierer für das SMS eingegeben werden, z.B.

```
Enter PIN for /usr/local/sms/pse:
Reenter PIN for /usr/local/sms/pse:
C=de,O=TUB,CN=SMS,SN=SYS00
```

Die SMS-Konfiguration muß entsprechend diesen Schlüsseln und der PIN angepaßt werden.

Als nächstes muß der öffentliche Schlüssel des SMS in eine Datei exportiert werden. Dies geschieht durch die Befehlsfolge:

```
secude> psemaint -p /usr/local/sms/pse
Enter PIN for pse:
PSE pse> read Cert sms.pub
```

Im aktuellem Verzeichnis entsteht nun die Datei “sms.pub”. Es bietet sich an, diese Datei unter /usr/local/sms/public-keys unterzubringen. Danach muß der öffentliche Schlüssel noch in die eigene Liste der öffentlichen Schlüssel eingetragen werden.

```
PSE pse> addpk Cert
```

Der Erfolg kann mit “show PKList” kontrolliert werden.

10.4 Installation eines SMG

Die SMG-Installation unterscheidet sich nur geringfügig von der SMS-Installation. Nur wird hier zwischen SMG und SMG-Send unterschieden. Wie beim SMS lege ich auch hier die Verzeichnisse bin/, etc/ und log/ an, diesmal aber im Oberverzeichnis /usr/local/smg.

ABBILDUNG 63. Die Datei smgrc

```
myname=C=DE,O=TUB,CN=SMG1,SN=SYS01
mypin=123
psepath=/usr/local/smg/pse
smglocalhost=smglocal.tu-berlin.de
smsaddress=sms.tu-berlin.de
smsname=C=DE,O=TUB,CN=SMS,SN=SYS00
sigHdrFailTxt=Beim ueberpruefen des Mailheaders ist ein Fehler aufgetreten.
sigBdyFailTxt=Beim ueberpruefen des Mailtextes ist ein Fehler aufgetreten.
secInfoTxt=Die E-Mail wurde sicher uebertragen.
```

Die Felder haben die gleiche Bedeutung wie bei der smsrc-Datei. Hinzu kommen die Felder:

- **smglocalhost** gibt an, wie der logische, lokale Name des Rechners lautet, der in der Mailtertable für den Mailer “local” eingetragen ist. Im Zweifelsfall kann das auch einfach die Bezeichnung “localhost” sein.
- **smsaddress** gibt den Rechnernamen an, der direkt zum SMS-Server führt.

10.4.1 Sendmail-Konfiguration

Für die Datei /etc/mailertable müssen diesmal zwei Einträge gemacht werden. Dabei ist “smg1” jeweils nur ein Beispielname. Jeder SMG bekommt seine laufende Nummer oder einen symbolischen Namen. Diese Namen werden in den Nameserverlisten für die Domäne angegeben. Im gegebenen Beispiel würde “smg1” als Rechner in der Liste “tu-berlin” geführt werden. Im ACM wird die Zuordnung von Personen zu Rechnern gemacht, indem die Attribute für die Mailadresse und den Schlüssel entsprechend dem zugehörigem Rechner gesetzt werden:

ABBILDUNG 64. Datei /etc/mailertable beim SMG

```
smglocal.tu-berlin.de   local:[127.0.0.1]
tc.tu-berlin.de         procmail:/usr/local/smgsend/etc/smgsendfilter
smg1.tu-berlin.de       procmail:/usr/local/smg/etc/smgfilter
```

Wieder ist der Rechner sowohl als smglocal.tu-berlin.de bekannt, als auch als smg1.tu-berlin.de. Der “local”-Eintrag dient der direkten Zustellung der E-Mail auf der Festplatte, die “procmail”-Varianten dienen dem Starten der jeweiligen Filterprogramme.

Sendmail sollte ferner so konfiguriert werden, daß es nur Mails vom SMS-Mailserver entgegennimmt, nicht als Relay verwendet werden kann und auch sonst keinen Zugang zuläßt.

10.4.2 Firewall-Konfiguration

Die Firewall kann auf diesem Rechner so eingerichtet werden, daß auf der Netzwerkschnittstelle zum Internet hin nur vom SMS auf den SMTP-Server (Port 25) zugegriffen werden kann und der SMG selbst DNS-Abfragen tätigen kann. Ferner muß die Firewall so konfiguriert werden, daß Mails an den SMS ausgeliefert werden können.

Der POP3-Port (110) darf nur für die Netzwerkkarte nach innen geöffnet werden.

10.4.3 Schlüsselverwaltung

Der Schlüssel wird beim SMG genauso generiert, wie beim SMS. Als Schlüssel-Identifizierer empfehle ich z.B. “C=DE,O=TUB,CN=SMG1,SN=SYS01”. Nach dem Exportieren des öffentlichen Schlüssels muß dieser dem SMS zugänglich gemacht werden. Dazu muß er auf den SMS-Server kopiert werden. Dort existiert ein Shellscript “addpk” im Verzeichnis /usr/local/sms/bin. Durch Starten des Programms mit dem öffentlichen Schlüssel als Parameter wird der Schlüssel der PKList im PSE des SMS hinzugefügt. Der öffentliche Schlüssel ist damit registriert.

Nun muß der öffentliche Schlüssel des SMS noch beim SMG registriert werden. Dazu muß die Datei mit dem öffentlichen Schlüssel auf den Rechner des SMG gebracht werden und dort analog zur Registrierung beim SMS mit dem Script “addpk” registriert werden.

Die registrierten Schlüssel können jederzeit mit dem Script “showpk” abgefragt werden. Jedes der Scripte fragt nach der PIN, um auf die PSE zugreifen zu können.

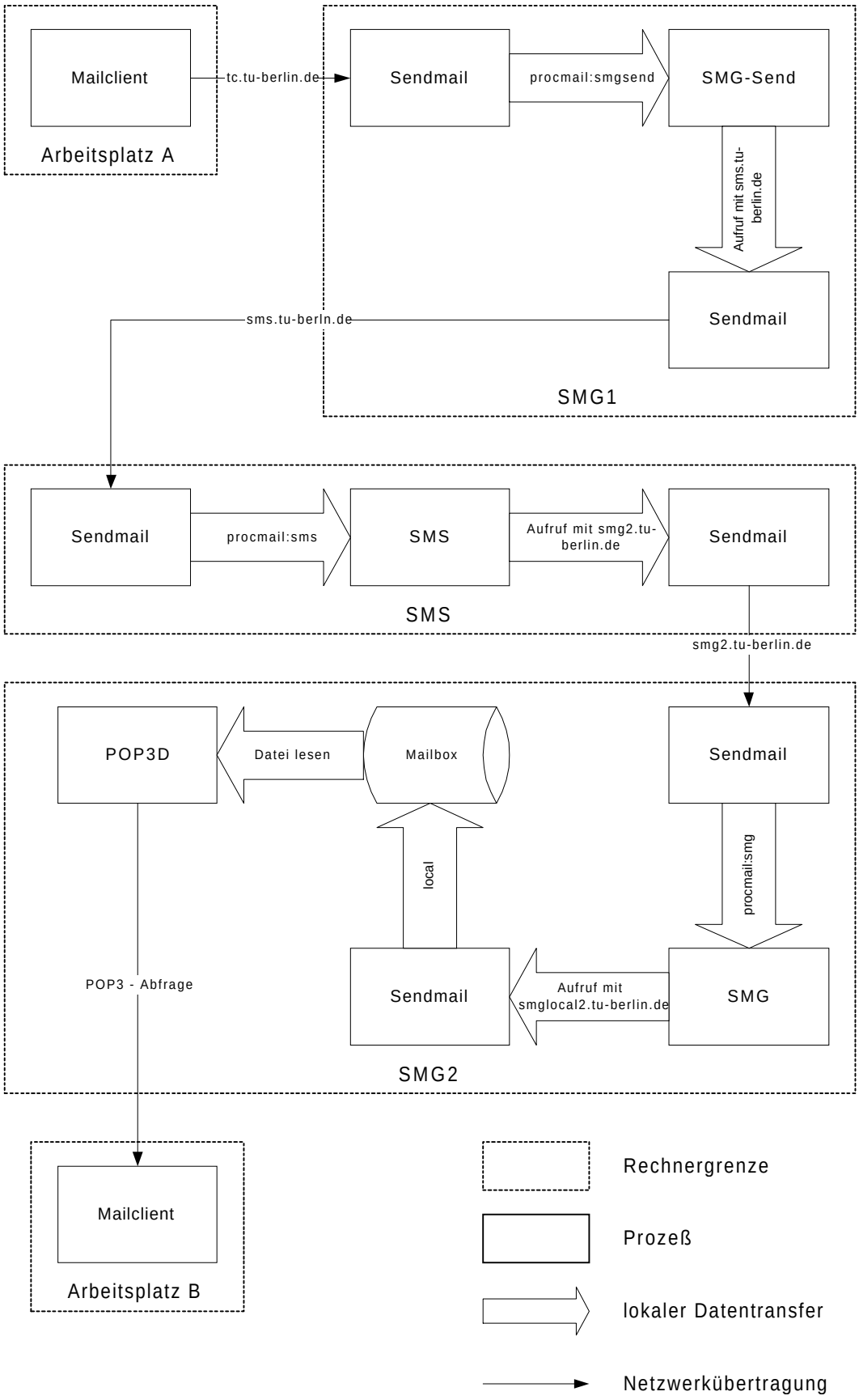
10.5 Von Adressen und Mailern

Um die Konfiguration der Mailertables besser verstehen zu können, hilft folgendes Diagramm (Abbildung 65 auf Seite 119). Es zeigt den Lauf der E-Mail durch die diversen Filter auf dem jeweiligen Rechner. Wichtig zum Verständnis ist dabei, daß die Bezeichnungen in der To:-Zeile im Mailheader jeweils gleich bleibt. Ich mache zum Weiterleiten der Mails davon Gebrauch, daß es möglich ist, die Mailerprogramme selbst einen Empfänger aufrufen zu lassen. Damit wird auf SMTP-Ebene das Data-Element nicht angerührt. Die Mail wird somit an einen anderen Empfänger ausgeliefert, als in der “To”-Zeile der E-Mail steht. Das ist kein besonderes Vorgehen. Die SMTP-Server arbeiten so z.B. auch, um Rechner zu erreichen, die nicht direkt angesprochen werden können, sondern hinter einem Gateway liegen. Auch Mailclient versenden ihre Mails über den Umweg eines bekannten SMTP-Servers zum eigentlichen System.

Die symbolischen Namen, wie z.B. “smglocal2” existieren nicht wirklich. Das dürfen sie auch nicht, weil es sonst möglich wäre, von außen E-Mails an diesen Rechner zu verschicken, die dann sofort zugestellt werden, ohne vorher durch die Filter gelaufen zu sein. Die symbolischen Namen, die bei mir den Suffix “local” tragen, sind Adressen, hinter denen die ARPA-Adresse 127.0.0.1 steht. Diese Adresse muß nicht über einen Nameserver aufgelöst werden können, sondern kann in der Datei /etc/hosts als Alias für “localhost” eingetragen werden.

Wenn es dem Administrator logischer erscheint, kann er auch gleich “localhost” in der smgrc-Konfigurationsdatei als lokalen Rechner angeben.

ABBILDUNG 65. Zustellung einer E-Mail über verschiedene Stationen



10.6 Installation der SPT auf einem PC

Im folgenden möchte ich nun die Installtion der Proxy-Tools unter Windows 95/98 und NT beschreiben. Hier kann man die zwei Schritte “Secude-Installation” und “SPT-Installation” unterscheiden.

10.6.1 Secude-Installation unter Windows 95/98

Für Secude ist es notwendig, daß bestimmte Einträge in der Windows Registrierung gemacht werden. Neuere Versionen von Secude erledigen diese Eintragungen über ein Installationsprogramm. Ferner müssen die DLL-Bibliotheken von Secude über die PATH-Variable erreichbar sein. Gewöhnlich wird das dadurch erreicht, indem die Programme ihre DLLs in das Windows-Verzeichnis kopieren. Hier gehe ich jedoch den Weg, die PATH-Variable zu erweitern.

- Von der SME Secude Installationsdiskette die Verzeichnisse SECUDE und GPK2000 auf Laufwerk C: kopieren.
- C:\AUTOEXEC.BAT editieren: PATH erweitern um ";C:\SECUDE;C:\GPK2000" (an bestehende PATH-Zeile anhängen).
- Aus Windows heraus START > Ausführen > "regedit". Unter dem Pfad HKEY_CURRENT_USER\SOFTWARE neuen Schlüssel hinzufügen mit der Bezeichnung "SECUDE". Neuen Schlüssel anwählen und Zeichenkette "psedir" hinzufügen. Wert dieser Zeichenkette ändern auf "c:\secude". Zweite Zeichenkette hinzufügen mit der Bezeichnung "etcdir". Wie oben Wert ändern auf "c:\secude\etc".
- Windows neu starten.

10.6.2 Secude-Installation unter Windows NT

Unter Windows NT gibt es keine PATH-Variable mehr in der AUTOEXEC.BAT. Das unter dem Windows 95/98 liegende DOS ist hier praktisch nicht mehr existent. Die Angabe der PATH-Variablen kann aus diesem Grund nicht über die Datei “autoexec.bat” erfolgen:

- Von der SME Secude Installationsdiskette die Verzeichnisse SECUDE und GPK2000 auf Laufwerk C: kopieren.
- Unter START > Einstellungen > Systemsteuerung > System > Umgebung die Umgebungsvariablen aufrufen. "PATH" editieren und um ";C:\SECUDE;C:\GPK2000" erweitern.
- Aus Windows heraus START > Ausführen > "regedit". Unter dem Pfad HKEY_CURRENT_USER\SOFTWARE neuen Schlüssel hinzufügen mit der Bezeichnung "SECUDE". Neuen Schlüssel anwählen und Zeichenkette "psedir" hinzufügen. Wert dieser Zeichenkette ändern auf "c:\secude". Zweite Zeichenkette hinzufügen mit der Bezeichnung "etcdir". Wie oben Wert ändern auf "c:\secude\etc".
- Windows neu starten.

10.6.3 Installation der SPT

Gegen die Secude-Installation ist die Installation des WINSPT relativ einfach. Nur ist es nicht üblich, unter Windows Benutzer Dateien editieren zu lassen. Hierfür müßte noch ein Installationsprogramm geschrieben werden, das diese Arbeit erledigt.

- Die beiden Dateien winspt.exe und winspt.ini in ein neues Verzeichnis C:\Programme\WinSPT kopieren (kann eigentlich auch jedes beliebige andere Verzeichnis sein).
- Mittels Start > Programme > Zubehör > Editor die winspt.ini anpassen. Das heißt in der Regel nur, den registrierten Schlüssel <name> unter myname=<name> ändern und ggf. die Variable mypin anpassen.

10.6.4 Schlüsselverwaltung

Grundsätzlich funktioniert die Schlüsselverwaltung unter Windows exakt genauso, wie beim SMG. Das Programm befindet sich hier unter C:\SECUDE\SECUDE.EXE und verhält sich genauso, wie sein UNIX-Pendant.

10.6.5 Sicherheitsanmerkungen zum PC

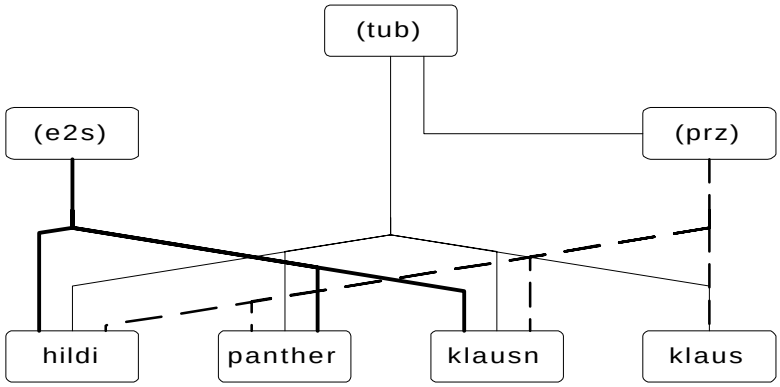
Es gibt diverse Diskussionen über die Sicherheit von Windows-PCs. Ferner bieten einige Firmen auch Software an, die den PC sicherer im Netzwerk machen sollen. Am besten versteckt sich ein Windows PC hinter einer Firewall. Da ich die SPTs jedoch gerade für den Fall konzipiert habe, daß keine Firewall bzw. SMG zur Verfügung steht, empfehle ich, die Schlüssel auf einer Smartcard unterzubringen. Somit kann selbst durch einen Angriff von außen keiner der Schlüssel entwendet werden. Mindestens sollten jedoch alle Dienste abgeschaltet sein, die einen Zugriff auf Drucker oder Dateien dieses PCs zur Verfügung stellen. Die relativ unsicheren PCs (wenn keine Smartcard benutzt wird), sind als solche bei Sicherheitsdiskussionen zu berücksichtigen. Sie sollten also als “nur zum Teil sicher” eingestuft werden.

10.7 Konfiguration des ACMI

10.7.1 Erstellung eines ACM mit E-Mail-Attributen

Wie schon diskutiert, arbeitet das SME mit dem ACM zusammen. Zum Betrieb eines SME muß also auch ein minimales AC-Modell erstellt werden. Ist das ACM ohnehin in Benutzung, kann das Modell für E-Mails mitbenutzt werden. Im Folgenden möchte ich an einer kleinen Beispielkonfiguration zeigen, wie man minimal ein ACM erstellen kann.

ABBILDUNG 66. Beispielkonfiguration eines ACM



Ich versuche hier einmal, eine Beispielkonfiguration dieser Struktur darzustellen. Die Entwurfschwäche der oben gezeigten Konfiguration ist, daß eine direkte Zuordnung zwischen Projekten und Personen sowie Instituten und Personen stattfindet. Idealerweise sollte eine solche Zuordnung immer über die Zwischenstufe Stelle gemacht werden (z.B. prz > wimi57 > klausn). Die Rollen sind in der Regel an die Stelle gebunden. Kündigt Klaus Nagel und wird seine Stelle neu besetzt, gehen auch alle seine Rollen und damit Rechte an seine Nachfolgerin oder seinen Nachfolger über.

Hier also nur zur Veranschaulichung Auszüge aus oben dargestelltem ACM:

```
User(
  Name: panther
  CreatorRole(
    Name: Creator
    Member: klaus
  )
  Attr(
    Name: arpa-email
    Type: mail
    Value: panther@smg1.tu-berlin.de
    ReadAllow: Public
    WriteAllow: Creator
  )
  Attr(
    Name: x400-name
    Type: mail
    Value: C=DE,O=TUB,CN=SMG1,SN=SYS01
    ReadAllow: Public
    WriteAllow: Creator
  )
  Attr(
    Name: Name
    Type: Description
    Value: J&ouml;rg Bartholdt
    ReadAllow: Public
    WriteAllow: Owner
  )
)
```

Es handelt sich bei diesem Objekt um ein “User”-Objekt, welches eine natürliche Person identifiziert. Dieses Objekt wurde von der Person “klaus” erstellt. Die Person besitzt die angegebene E-Mailadresse. Diese darf vom Ersteller geändert werden. Zu dieser E-Mailadresse gehört der Schlüssel mit der unter “x400-name” angegebenen Identifikation. Ferner besitzt das Objekt noch eine Beschreibung. Diese Beschreibung lautet “Jörg Bartholdt” (in HTML-Umschreibung).

Jörg Bartholdt sitzt also aus Sicht des SMS hinter dem SMG1 und hat deshalb auch keinen eigenen Schlüssel, sondern nur den Sammelschlüssel des SMG1.

```
User(
  Name: hildi
  CreatorRole(
    Name: Creator
    Member: klaus
  )
  Attr(
    Name: arpa-email
    Type: mail
    Value: hildi@smslocal
    ReadAllow: Public
    WriteAllow: Creator
  )
  Attr(
    Name: x400-name
    Type: mail
    Value: C=DE,O=TUB,CN=Thomas.Hildmann,SN=2834728
    ReadAllow: Public
    WriteAllow: Creator
  )
  Attr(
    Name: Name
    Type: Description
    Value: Thomas Hildmann
    ReadAllow: Public
    WriteAllow: Owner
  )
)
```

Thomas Hildmann ist Besitzer eines SPT-Rechners. Ihm ist ein eigener Schlüssel zugeordnet. Ferner wird auf dem SMS die E-Mail lokal zugestellt. Thomas Hildmann wird seine E-Mails dann über POP3 direkt vom SMS abfragen.

Neben den Benutzern gibt es die “Service Objekte”. Diese habe ich im Diagramm durch Einklammerung der Namen gekennzeichnet. Diese beschreiben ordnende Objekte, an die vor allem Rollen geknüpft sind.

```
Object(
  Name: e2s
  CreatorRole(
    Name: Creator
    Member: klaus
```

```
  )
  AllRole(
    Name: All
  )
  Role(
    Name: Leiter
    Member: klausn
  )
  Role(
    Name: Mitarbeiter
    Member: klausn panther hildi
  )
)
```

Das sehr vereinfachte Objekt “e2s” umfaßt die Rollen “Leiter” und “Mitarbeiter”. Es existiert immer auch eine Rolle “All”, die dann vom ACMI angenommen wird, wenn keine Rolle bei einem Objekt spezifiziert ist. Es ist also die Standardrolle (engl. default role).

Die beschriebene Methode “resolveAddr” läßt sich genau genommen nur auf Rollen anwenden. Versucht man also, ein “resolveAddr” auf “e2s” zu machen, wird in Wirklichkeit “e2s.All” angenommen und dieses aufgelöst. Ferner ist es, wie gerade spezifiziert, möglich, “e2s.Mitarbeiter” aufzulösen oder “e2s.Leiter”.

Ich spare mir an dieser Stelle, die Spezifikation der anderen Objekte hier abzudrucken. Das genaue Vorgehen beim Erstellen eines ACM kann bei [Barth98] nachgelesen werden. Die Spezifikation der anderen Objekte läuft analog zu den eben gezeigten.

Im geringsten Fall reicht es aus, ein Modell zu erstellen, das nur aus User-Objekten besteht. Es können aber auch beliebig viele Rollen und Service-Objekte enthalten sein. Jedes “resolveAddr” löst das Netzwerk von Objekten so weit auf, daß jede verknüpfte Adresse genau einmal zurückgemeldet wird.

10.7.2 Testen der ACM-Konfiguration

Die Konfiguration kann schließlich getestet werden, indem man das ACMI mit dem Modell startet und ein “telnet localhost 3000” durchführt. Der folgende Auszug zeigt einige Beispielabfragen des beschriebenen Modells:

- Die folgende Abfrage simuliert ein Rundschreiben an alle Personen, die etwas mit der TUB zu tun haben.

```
Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Target: tub

Protocol: TCACM/long
User: hildi
Object: /
```

```
Method: resolveAddr
Return: Success
klaus@smg2.tu-berlin.de, C=DE,O=TUB,CN=SMG2,SN=SYS02
klausn@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
panther@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
hildi@smslocal, C=DE,O=TUB,CN=Thomas.Hildmann,SN=2834728
```

- Diese Abfrage simuliert das Auflösen von genau einer Benutzeradresse mit dem dazugehörigen Schlüssel.

```
Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Target: klausn

Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Return: Success
klausn@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
```

- Hier wird abgefragt, wer alles Mitarbeiter beim PRZ ist.

```
Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Target: prz.Mitarbeiter

Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Return: Success
klaus@smg2.tu-berlin.de, C=DE,O=TUB,CN=SMG2,SN=SYS02
klausn@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
panther@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
hildi@smslocal, C=DE,O=TUB,CN=Thomas.Hildmann,SN=2834728
```

- Hier wird explizit an alle Rollen im E2S-Team geschrieben. Dies ist identisch mit der Angabe:

```
Target: e2s
```

```
Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Target: e2s.All

Protocol: TCACM/long
User: hildi
Object: /
Method: resolveAddr
Return: Success
klausn@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
panther@smg1.tu-berlin.de, C=DE,O=TUB,CN=SMG1,SN=SYS01
hildi@smslocal, C=DE,O=TUB,CN=Thomas.Hildmann,SN=2834728
```

Die Angaben der Mailadressen sehen entsprechend aus, nur daß hier noch die Angabe des logischen Namens des SMS angehängt wird. In unseren Szenarien beschreiben wir dieses immer als “tc.tu-berlin.de”.

So könnte eine E-Mail an “prz.Leiter@tc.tu-berlin.de” geschrieben werden. Ist die Logik der Adressierung erst einmal ausreichend bei den Benutzern bekannt, ist es praktisch nicht mehr erforderlich, die Adressen wirklich zu lernen. Sie können intuitiv selbst “geraten” werden z.B. “prz.Sekretariat@tc.tu-berlin.de” oder “tub.StudentischeHilfskraefte@tc.tu-berlin.de”.

KAPITEL 11

Validierungsszenarien

Die Validierung des SME Systems ist recht problematisch. In dieser Arbeit ging es nicht darum, Verschlüsselungsverfahren zu implementieren. Diese könnte man z.B. über ihre mathematische Korrektheit validieren. Hier geht es darum, Konzepte zu validieren. Und üblicherweise lassen sich solche Konzepte nur über eine genaue Diskussion untersuchen.

Ich möchte die Validierung in zwei große Teile gliedern. Zum einen ist dort die Funktionalität der Mailverteilung und Auslieferung, zum anderen ist das Datenformat zu betrachten, also Angriffspunkte, die durch Fehler in der Datenrepräsentation auftreten können.

Wenn das ACM korrekt arbeitet, die Kommunikation mit dem ACM korrekt implementiert ist, und jeweils auch der gültige Schlüssel ermittelt und benutzt wird, ist der gesamte Teil korrekt.

An dieser Stelle ist das SME vor allem auf das Zusammenspiel der externen Komponenten angewiesen. Ich möchte diesen Punkt nicht unterbewerten. Er kann jedoch mit einer ausreichend großen Zahl an Tests geprüft werden.

Im Vordergrund dieser Validierung soll das Sicherheitskonzept des SME stehen, da dies ein Punkt ist, der sich nur schwer über Tests prüfen lassen wird. Es gibt kein universelles Angriffsmodell, mit dem man prüfen könnte, ob es möglich ist, die Sicherheit der übertragenen E-Mails zu verletzen.

Man kann sich bei solchen Sicherheitsprodukten auch nie sicher sein, ob es nicht irgendwann einen Angriffspunkt gibt, der zur Zeit nur einfach noch nicht bekannt ist. Mit anderen Worten: Es könnte irgendwann (in naher Zukunft) eine Angriffsmöglichkeit geben, die heute noch nicht bekannt und die einfach aus diesem Grund nicht bedacht ist.

Im folgenden möchte ich die Sicherheit des SME nach meinem Wissensstand der möglichen Angriffsformen untersuchen und diskutieren.

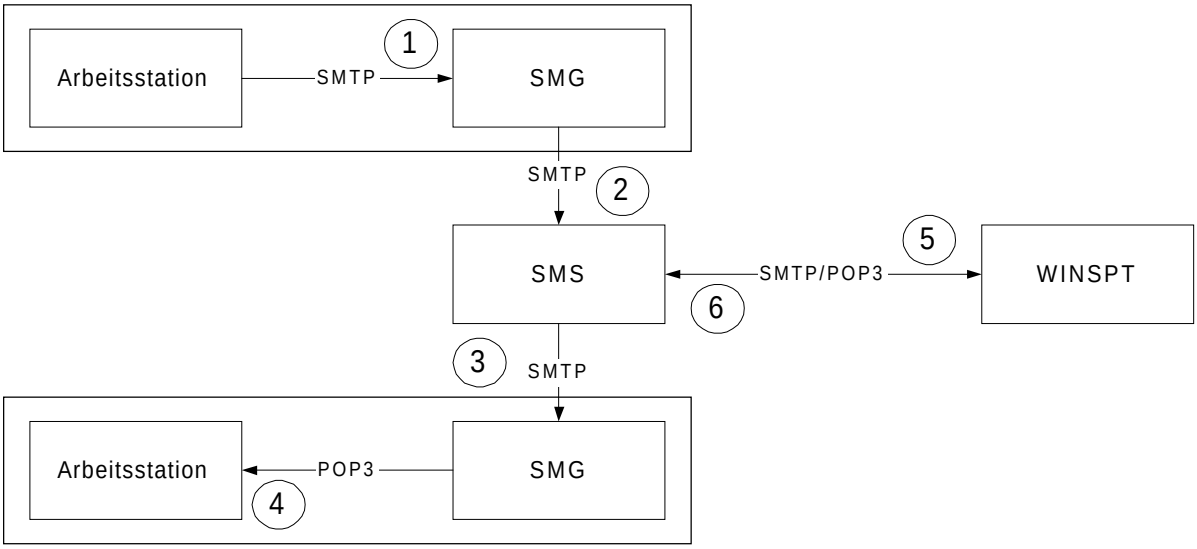
Ich möchte dabei nach folgender Systematik vorgehen: Ein beliebter Angriffspunkt sind die Kommunikationswege in einem System. Bei der Übertragung der Daten (insbesondere über ein Netzwerk) bieten sich die einfachsten Möglichkeiten der Datenmanipulation oder der Spionage. Aber auch die Datenrepräsentation kann ein Angriffspunkt sein. An letzter Stelle stehen selbstverständlich auch Angriffe auf die Programme oder die Maschinen mit ihren Betriebssystemen.

11.1 Untersuchung der Kommunikationswege

Ich möchte nun alle möglichen Kommunikationswege unter den SME-Komponenten zusammenstellen, um darauf aufbauend analysieren zu können, ob die jeweilige Kommunikation ein Sicherheitsrisiko darstellt.

Die Komponenten sind bereits bekannt. Das folgende Diagramm zeigt alle möglichen Kommunikationswege. Dabei wählte ich nur aus Darstellungsgründen zwei SMGs aus.

ABBILDUNG 67. Mögliche SME Netzwerkverbindungen



Jeder Pfeil repräsentiert eine TCP/IP Socketverbindung über eine Netzwerkleitung. Die Kästen repräsentieren jeweils Rechner mit den bekannten Bezeichnungen. Die Umrahmung von Arbeitsstation und SMG soll symbolisieren, daß es sich hier um einen als sicher zu betrachtenden LAN Netzabschnitt handelt. Zur einfacheren Benennung habe ich die Netzwerkleitungen durchnumeriert.

(1) Die Arbeitsstation sendet ihre Daten über das SMTP Protokoll an den SMG Rechner. Hierbei führt die Datenkommunikation nur über den LAN Abschnitt, der als sicher angenommen wurde.

(2) Der SMG sendet die verschlüsselte E-Mail an den SMS. Die Kommunikation führt über einen ungesicherten Netzabschnitt. Da die E-Mail verschlüsselt und signiert ist, kann sie weder unbemerkt verändert noch gelesen werden. Sie kann jedoch auf dem Weg zerstört werden, so daß sie nie den SMS erreicht. Bei einer verwaltungstechnischen Umsetzung ist darauf Rücksicht zu nehmen, daß das Ausliefern der E-Mails nicht garantiert werden kann. So müßte der Erhalt einer E-Mail z.B. bestätigt werden. Zwar führen “Sendmail” und “POP3d” Protokolle über das Versenden und Empfangen von E-Mails, diese Protokolle können jedoch auch gefälscht werden, beinhalten nur sehr wage Informationen über die E-Mail und können in der Regel auch nur innerhalb eines begrenzten Zeitraums per Hand ausgewertet werden.

(3) Der SMS sendet die umverschlüsselte E-Mail an den SMG. Auch diese Kommunikation läuft über eine “unsichere” Leitung. Die E-Mail ist jedoch verschlüsselt und signiert und kann im schlimmsten Fall auf dem Weg vernichtet werden.

(4) Die Arbeitsstation fragt die E-Mail über das POP3-Protokoll ab. Die Firewallkonfiguration läßt nur eine Abfrage von innerhalb des “sicheren” LANs zu. Aus diesem Grund kann diese Kommunikation auch als “sicher” angenommen werden.

(5) Ein mit einem WINSPT ausgestatteter Windows PC fragt E-Mails über das “unsichere” Netz ab. Als Protokoll dient hier POP3. Die E-Mail liegt verschlüsselt und signiert auf dem SMS. Sie kann auf

dem Weg also nicht verändert oder gelesen werden. Das POP3 Protokoll ist hier das größte Problem. Zur Authentifizierung benutzt es unverschlüsselte Paßwörter. Deshalb muß damit gerechnet werden, daß Angreifer Paßwörter mittels Abhören von Netzwerkverbindungen (sog. Sniffing) ausspionieren und so z.B. einfach Mails abfragen kann, als wäre er der eigentliche Benutzer. Das bringt den Angreifer keinen besonderen Vorteil, was den Inhalt der Mails angeht. Damit ist jedoch nicht mehr nachweisbar, ob ein SPT-Benutzer eine E-Mail jemals erhalten hat oder nicht.

(6) Das Senden über das WINSPT ist unproblematisch. Hier stellt sich die gleiche Situation, wie bei (2) dar.

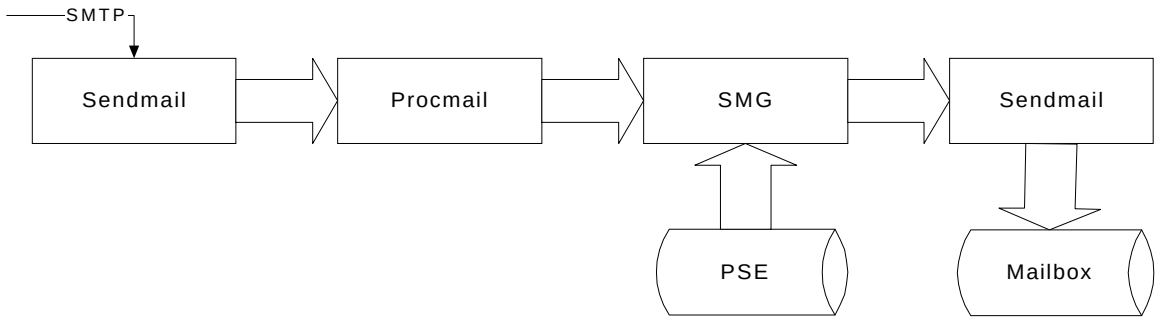
Ich untersuche nun die Kommunikation innerhalb der Rechner, also die Datenübertragung mittels Inter-Prozeß-Kommunikation (engl. Inter Process Communication, IPC genannt).

11.1.1 Betrachtung der IPC im SMG

Auch die Datenübergabe zwischen den einzelnen Modulen und die verwendeten Dateien können Ziel einer Attacke sein. Im folgenden Diagramm sind neben den bekannten Softwaremodulen auch die Dateien eingezeichnet, in die die Module schreiben und aus denen sie lesen.

Dabei ist Procmail nur der Interpreter, der das Skript “smgfilter” ausführt. “SMG” steht für die ausführbare Datei “smg.e”. PSE bezeichnet im Diagramm die Software PSE, die von Secude angelegt wurde. “Mailbox” bezeichnet die Datei, in die die E-Mails des jeweiligen Empfängers einsortiert werden. In der Regel geschieht das unter LINUX im Verzeichnis “/var/spool/mail” oder “/var/mail”.

ABBILDUNG 68. Datenaustausch beim Empfangen im SMG



Die Komponente Sendmail ist ein sicherheitstechnisch viel diskutiertes Programm. Da das Programm mit Superuser-Rechten auf dem UNIX-System laufen muß, um alle Aufgaben zu erfüllen, ist es ein beliebtes Angriffsziel. Viele Sicherheitsdiskussionen und entsprechende Korrekturen an Sendmail zeigen das. Ich bin jedoch der Überzeugung, daß die aktuelle Sendmailversion ein kalkulierbares Risiko darstellt. Die Administratoren der Rechnersysteme werden jedoch auch in Zukunft die Diskussionen um evtl. vorhandene Sicherheitslücken weiter verfolgen müssen. Es ist aber immernoch sicherer, das vorhandene Sendmail zu benutzen, als ein Produkt dieser Art neu zu erstellen.

Procmail wird unmittelbar von Sendmail aufgerufen. Die E-Mail wird mittels des UNIX-Pipe-Mechanismus als Standardeingabe an Procmail übergeben. Hier findet die Auswertung von Regeln statt, die nur eine Auslieferung über das SMG-Programm zuläßt. Zu diesem Zeitpunkt ist die E-Mail noch verschlüsselt. Es hätte für einen Angreifer also noch nicht einmal Sinn, ein falsches Procmail einzuschleusen. Es sei denn, es wird dazu benutzt, um die Mail wieder aus dem abgesicherten LAN herauszubefördern. Diese Möglichkeit sollte deshalb über die Firewall untersagt sein. Damit decken wir jedoch eine Lücke auf, die durch die Anforderung entsteht, auch unverschlüsselt Mails verschikken zu können. Durch das Voranstellen eines “_”-Zeichens (Unterstrichs) soll dem SMG-Send signalisiert werden, daß die E-Mail im unverschlüsselten Zustand das SME verläßt oder unverschlüsselt im

SME zugestellt wird. An dieser Stelle greift jedoch wieder die Argumentation, daß das LAN selbst als “sicher” anzusehen ist. Die Voraussetzung war ja, daß angenommen wird, daß Angriffe aus dem LAN selbst nicht stattfinden. Gelingt einem Angreifer jedoch ein Zugriff auf den SMG-Rechner, so ist er mindestens in der Lage, im Namen anderer E-Mails zu verschicken. Erlangt dieser Angreifer auch Superuser-Rechte, so kann er E-Mails auch nach außen befördern.

Weiterer Angriffspunkt ist die PSE. Wird diese gestohlen, kann im Namen des SMG gehandelt werden. Das heißt, es könnten rechtsgültige Unterschriften im Namen der Gruppe hinter dem SMG gemacht werden. Dabei ist der Schutz durch die PIN nicht besonders hoch. Es dauert nur Stunden, um eine unbekannte PIN einer Software-PSE durch eine Brute-Force-Attacke zu ermitteln. Dieser Zeitraum könnte bei rechtzeitiger Entdeckung dazu genutzt werden, einen neuen Schlüssel zu generieren und den alten als ungültig zu erklären.

Das SMG-Programm selbst könnte vor allem durch Überlastungen o.ä. angegriffen werden. Dieses Problem ist jedoch nicht als besonders hoch einzustufen, da, wie bereits beschrieben, sog. Secondary Mailserver Mails aufnehmen können, wenn das SMG überlastet ist. Dieser Mechanismus ist schon lange im Einsatz und funktioniert auch bei Angriffen erstaunlich gut. Das gilt auch dann, wenn der Angreifer versucht, die Festplatte des SMG vollaufen zu lassen, indem er Unmengen an Maildaten an das SMG schickt. Die Festplatte kann zwar gefüllt werden, Sendmail nimmt aber dann keine Daten mehr an und der Secondary Mailserver übernimmt das Aufspeichern der Mails, bis auch dieser voll ist.

Irgendwann kann jeder Platz aufgebraucht werden. Hierfür sind geeignete Maßnahmen von seiten der Rechnerwartung zu treffen. Diese müssen auf die Bandbreite und den zur Verfügung stehenden Platz abgestimmt werden.

Ein großes Problem stellen die Permissions der PSE dar. Sendmail wechselte seine effektive Benutzerkennung bei der Zustellung von E-Mails auf die des Empfängers. Damit können Programme von Sendmail für den Benutzer ausgeführt werden, ohne daß die Gefahr besteht, daß der Benutzer sich damit unrechtmäßigen Zugang zu sonst verbotenen Funktionen beschafft.

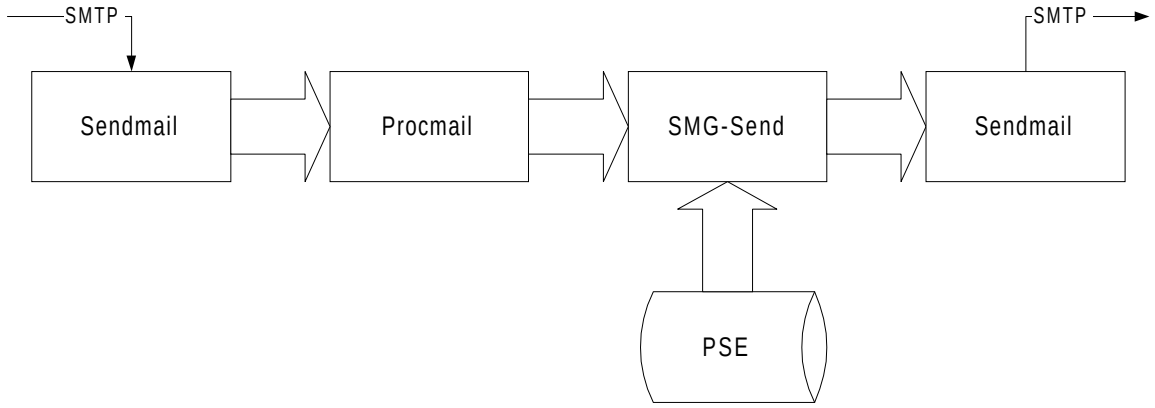
Das heißt: Wenn eine E-Mail für den Benutzer “klausn” ausgeliefert wird, läuft auch der Sendmailprozeß, damit auch der Procmail und schließlich auch der “smg.e” unter der Benutzerkennung “klausn”. Die einzige Möglichkeit, trotzdem kryptographisch arbeiten zu können, ist allen Benutzern Zugang zur PSE zu gewähren, inklusive der Rechte auf den privaten Schlüssel.

Ich habe zwei Antworten auf diese Sicherheitsfrage:

1. Man kann den SMG-Prozeß mittels “Set-UID-Flag” immer unter einer Kennung, wie z.B. “sme” laufen lassen. Die PSE müßte dann nur “sme” zugänglich sein. Da der SMG-Prozeß selbst wieder ein Sendmail startet, wird dieses wiederum mit der Benutzerkennung des Empfängers laufen. Intern wird die E-Mail mittels IPC vom gestarteten Sendmail-Prozeß an den Daemon-Prozeß, der ständig läuft, übertragen und von diesem Prozeß, der unter Superuser-Rechten läuft, ein Prozeß abgespalten (mittels fork()), der die effektive Benutzerkennung des Empfängers hat.

2. Gibt man den Benutzern des LANs keinen Zugriff auf den SMG-Rechner, sondern trägt die Benutzer lediglich mit einer Shell, z.B. “/usr/bin/true” ein, und ordnet ihm ein nicht existentes Heimatverzeichnis zu, so kann die PSE auch so nicht erreicht werden. Der SMG sollte sowieso für nichts anderes eingesetzt werden, als zu Firewall- und Gateway-Zwecken (evtl. noch als WWW-Proxy o.ä.).

ABBILDUNG 69. Datenaustausch beim Senden im SMG



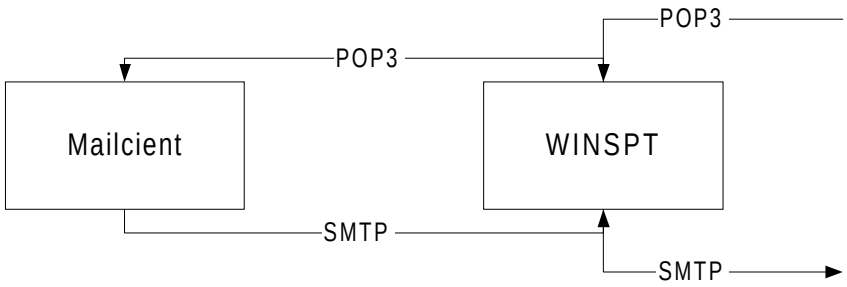
Beim Senden stellt sich das Problem sehr ähnlich wie beim Empfangen dar. Der Unterschied ist hier, daß es das Problem mit den Zugriffsrechten auf das PSE nicht gibt, da hier alles unter einer effektiven Benutzerkennung läuft.

11.1.2 Betrachtung der internen Kommunikation bei den SPTs

Das Programm WINSPT dient als Proxy. Aus Darstellungsgründen habe ich in der Grafik die PSE weggelassen, auf die das WINSPT zugreift. Wie wir sehen, gibt es hier intern zwei Socket-Verbindungen, die als Angriffspunkt genutzt werden könnten. Ein solcher Angriff könnte wie folgt aussehen:

Ein Angreifer stellt fest, daß ein PC mit einem SPT ausgestattet ist. Er trägt in seinem Mailer die IP-Nummer des SPT-Rechners und den Port 25 ein. Jetzt erstellt der Rechner des Angreifers eine Verbindung zum Proxy her und dieser arbeitet, als würde die Mail vom lokalen Mailclient kommen. Der Proxy dient also dem falschen Client.

ABBILDUNG 70. Netzwerkverbindungen bei den SPTs



Um diesen Angriff zu verhindern, muß der WINSPT die Verbindung auf die IP-Nummer 127.0.0.1 (eine lokale Verbindung) beschränken.

Die andere Angriffsmöglichkeit bezieht sich auf die PSE, die auf der Festplatte des PCs gespeichert ist. Es gibt diverse Programme, über die sich ein Angreifer Zugang zu diesen Dateien beschaffen könnte. Aus diesem Grund sollte von fachkundigem Personal mindestens eine Liste von Programmen aufgestellt werden, die relativ bedenkenlos auf den PCs installiert werden dürfen und die die Sicherheit nicht unnötig gefährden.

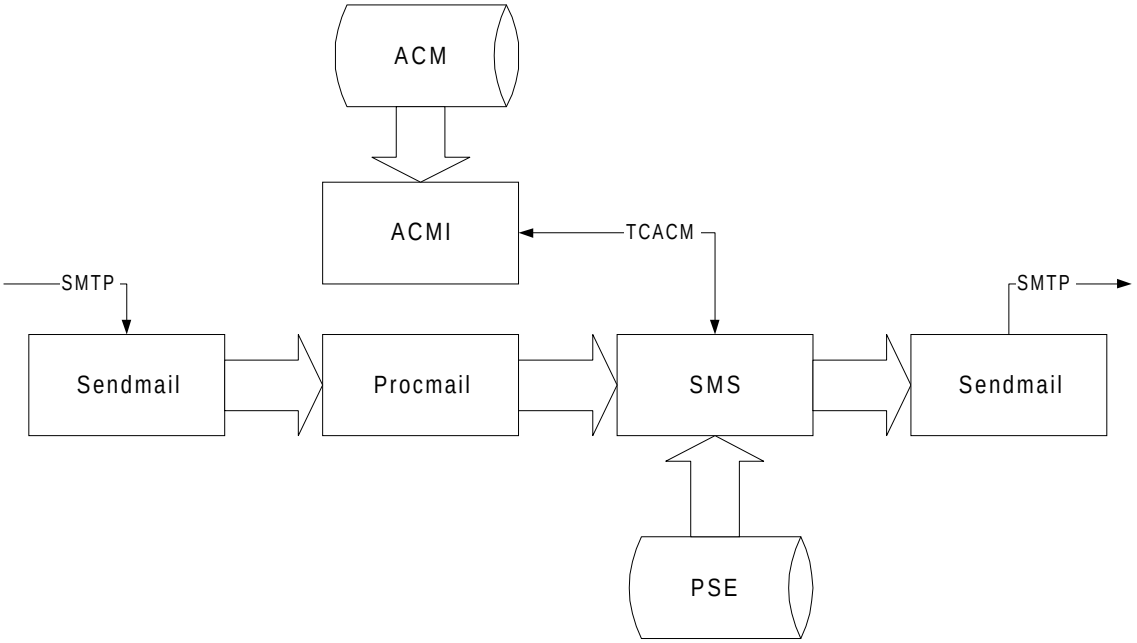
Ich plädiere jedoch immer für die Verwendung einer Smartcard in dieser Konfiguration.

11.1.3 Datenaustausch innerhalb des SMS

Der SMS wird der zentrale Angriffspunkt sein. Hier kann im Falle einer Attacke der größte Schaden angerichtet werden, da nicht nur ein System sondern gleich alle Systeme betroffen sind.

Der SMS-Rechner (im Idealfall eine CMW-Maschine) sollte schon räumlich gut geschützt aufbewahrt werden, da sich hier auch ein physikalischer Zugriff lohnen könnte, um beispielsweise den Organisationsschlüssel zu erlangen.

ABBILDUNG 71. Kommunikation innerhalb des SMS



Wie man auf diesem Diagramm sieht, besitzt das SMS die gleichen Angriffsmöglichkeiten wie das SMG. Hier gibt es auch das Problem mit dem Schutz der PSE.

Hinzu kommt als Angriffspunkt das ACMI und die Datei mit dem Modell (im Diagramm als “ACM” dargestellt). Die Systembetreiber müssen dafür Sorge tragen, daß kein Benutzer Prozesse auf dieser Maschine starten kann. Es wäre sonst denkbar, daß ein Benutzer herausfindet, wie man den ACMI zum Absturz bringen kann und einen eigenen ACMI startet. Dieser könnte dann ein Modell beinhalten, das ihm von jeder E-Mail eine für ihn verschlüsselte Kopie zustellt.

Diese Möglichkeit ist auch im Konzept des “seperation-of-duty” nicht ausgeschlossen. Für eine wirkliche Trennung müßte das ACMI und das ACM wieder in einem eigenen Compartment der CMW-Maschine laufen.

11.2 Betrachtung der Datenrepräsentationen

Ich möchte an dieser Stelle keine Bewertung des PEM-Standards durchführen. Hierzu finden sich Papiere verschiedenster Experten (z.B. [Schn96]).

Für mich ist jedoch die Frage interessant, wie sicher die Header-Signatur wirklich ist. Betrachten wir noch einmal das Prinzip: Der normalisierte Header wird unterschrieben. Soweit besteht kein Problem. Es dürfte nahezu unmöglich sein, einen zweiten normalisierten Header zu schaffen, der bei gegebener

Signatur gültig ist. Damit aber der Header nicht vom Body getrennt werden kann, muß noch ein Hashwert über den Body berechnet und mit in die Signatur aufgenommen werden. Ansonsten wäre z.B. folgender Angriff möglich: Alice schickt eine E-Mail an Bob. Carol fertigt sich bei der Übertragung zwischen SMG und SMS eine Kopie der verschlüsselten E-Mail an. Jetzt trennt Carol den Body vom Header und ersetzt den Header durch einen Header einer E-Mail, die sie irgendwann erhalten hat. Beim erneuten Versenden an das SMS wird Carol der Body der E-Mail an Bob an sie adressiert und für sie verschlüsselt zugesandt. Sie ist nun in der Lage, Bobs E-Mail zu lesen.

PEM hat die Angewohnheit, die Eingaben zu prüfen, die zwischen der Start- und der Endmarke einer Nachricht stehen (siehe Beispiel in „Header-Signaturen“ auf Seite 28). Die Daten davor und danach werden ignoriert. Ein Angreifer könnte nun seinen eigenen Body zusammenstellen und an den falschen Header hängen. Damit aber die Signatur im Header stimmt, fügt er solange zufällige Zeichen an die Mail an, bis der berechnete Hashwert wieder stimmt. Die zufällig angehängten Zeichen gehen dann beim Entschlüsseln der E-Mail verloren. Außerdem können sie dem Angreifer egal sein, wenn er die Mail an sich selbst schickt. Damit wäre es möglich, eine beliebige E-Mail abzufangen und mit Hilfe eines Headers, einer Mail, die an den Angreifer ging, erneut an den Angreifer zu senden. Um dies zu verhindern, muß entweder der Hashwert aus der Signatur des Bodytextes gewonnen und verwendet werden oder die Überprüfungsroutinen so intelligent gestaltet werden, daß sie mit diesem Trick nicht zu täuschen sind.

11.3 Mögliche Angriffe auf die Rechner

Jedes System bleibt für sich natürlich allein durch seine Dienste nach außen angreifbar. Beispiele hierfür sind Angriffe, die gegen IMAP-Server auf Linux-Rechnern durchgeführt wurden oder Fehler in SSH-Servern. Aus diesem Grund empfehle ich, die SMG-Rechner und ganz besonders den SMS-Rechner nur direkt an der Konsole zu warten und alle Dienste außer SMTP und POP3 auf diesen Maschinen abzuschalten. Damit beschränken sich die Angriffsmöglichkeiten genau auf die zur Verfügung gestellten Dienste. Das ist das kalkulierte Restrisiko, das die Organisation tragen muß.

Sind die genannten Angriffschancen zu groß, kann die Antwort nur lauten, daß das System nicht eingesetzt werden kann. Eine solche Restrisiko-Abwägung muß vor Einführung gemacht werden.

11.4 Bekanntwerden von Schlüsseln

Wird ein Schlüssel einer Mitarbeiterin oder eines Mitarbeiters bekannt, so ist es vor allem wichtig, daß dies der Organisation mitgeteilt wird, sofern dies überhaupt entdeckt wird. Der Verlust eines Schlüssels im SME stellt keinen besonders großen Schaden dar, solange er gemeldet wird. Sofort kann im SMS und im SMG oder auf dem SPT-Rechner das Schlüsselpaar ausgetauscht bzw. neu registriert werden.

Schwieriger gestaltet sich der Verlust des SMS-Schlüssels. Ein solcher Verlust muß den vollständigen Neuaufbau der gesamten Schlüssel-Infrastruktur mit sich ziehen. Es muß ein neuer SMS-Schlüssel generiert werden. Der öffentliche Teil des Schlüssels muß an alle SMGs und SPTs gegeben werden.

KAPITEL 12

Aussichten

Die Resonanz, die meine Arbeit im Projekt brachte, und die vielen Leute, die mich immer wieder gefragt haben, wann meine Arbeit denn nun endlich fertig sei zeigen, daß an einem System wie diesem durchaus ein Interesse besteht, welches es sinnvoll macht, den Weg weiter zu verfolgen.

Wie so häufig bei Softwareprojekten wurde auch ich von neuen Entwicklungen überrannt. Mit nur einer Person läßt sich kaum mit den wichtigsten Entwicklungen Schritt halten. Das SME ist in der Form, wie es heute existiert, im Grunde veraltet. Aber die TU-Verwaltung schreibt noch immer ihre Rundschreiben per Papier oder Fax, noch immer stören Anrufe, die eigentlich nur Anfragen darstellen, die schon längst über ein sinnvolles EDV-System gelöst sein müßten oder die sich auch schnell per E-Mail klären lassen.

E-Mail ist im Projektleben an der TU Alltag. Das sollte es aber auch in der Verwaltung sein. Und nicht nur hier, sondern an vielen anderen Stellen auch. Während meiner Präsentation beim 5. Workshop "Sicherheit in vernetzten Systemen" beim DFN-CERT in Hamburg am 4. und 5. März diesen Jahres, lernte ich einige Hochschullehrer kennen, die ebenfalls an einem System wie dem SME interessiert waren. Leider hatte zu diesem Zeitpunkt die Software noch lange keinen Stand erreicht, den man aus der Hand geben konnte. Und wie es so häufig ist: Das Produkt braucht Tester, wie ich bereits schon geschrieben habe.

Es muß erst einmal genügend Menschen geben, die sich mit der Idee des SME vertraut gemacht haben, um das Potential zu begreifen, das in dieser Lösung steckt. Dabei ist der Sicherheitsaspekt nur ein kleiner Teil. Viel interessanter sind die Möglichkeiten, die aus der Kombination des rollenbasierten Zugriffssystems (ACM) und der E-Mail entstehen. Die sichere Übertragung sehe ich in diesem Zusammenhang nur als Ausbesserung. Da wir auf das Internet zurückgreifen, das nie als "sicheres Netzwerk" konzipiert wurde, brauchen wir Hilfsmittel, wie SMGs oder SPTs, um die Mankos auszugleichen.

Vielleicht ist das Konzept aber für ganz andere Entwicklungen gut. Vielleicht bleiben Verschlüsselungsprogramme doch immer Module, die in Mailclients integriert werden und ihre Schlüssel bei Zertifizierungsstellen anfordern. Die Versorgung von Mailinglisten über eine Umverschlüsselung und die Signatur des Headers wären hier noch immer ein Ansatz für die Durchführbarkeit solcher Vorhaben.

Wer weiß, vielleicht endet das Konzept des SME ja irgendwann als Vorlage für ein intelligentes Mailinglisten-Verwaltungs-System, das auch moderneren Ansprüchen gerecht werden kann.

Vielleicht wird das SME aber auch genau für die Anwendungsfälle eingesetzt, für die es einst konzipiert wurde.

Die Einführung von PGP 5.0 in der Firma, in der ich zur Zeit arbeite, hat mir gezeigt, daß selbst diese Lösung weit davon entfernt ist, von Menschen eingesetzt zu werden, die sich mit der Materie noch nicht eingehend befaßt haben. Ein Ansatz wäre nun zu behaupten: Dann müssen die sich eben damit

befassen. Viele Sicherheitsexperten, die ich kennengelernt habe, argumentieren genau so. Mein Ansatz hingegen ist jedoch, daß die Benutzer gar nichts von der “sicheren E-Mail” mitbekommen müssen. Was genügt, ist das Wissen darüber, welche E-Mail vertrauenswürdig ist und welche nicht. Das SME geht genau diesen Weg.

Aus diesem Grund ist es meine Überzeugung, daß es genau für dieses System, mit diesen Eigenschaften Abnehmer gibt. Ist das SME erst einmal installiert, bedarf es auch keiner weiteren Pflege. Ein Systemadministrator ist in der Lage, die Verwaltung für eine relativ große Belegschaft zu gewährleisten.

Ich wäre sehr froh darüber, wenn das SME Konzept eine Anwendung finden würde, in welcher Form auch immer. Ich würde mich auch freuen, wenn ich weiter an der Entwicklung dieses Systems teilhaben könnte. Vielleicht finden sich auch andere Interessenten, die mit mir zusammen die Zeit finden, auf Basis der Überlegungen des SME ein Produkt zu entwickeln, das entweder kommerziell vermarktet wird oder als GNU General Public Licence (GNU-GPL) Produkt erscheinen kann.

12.1 Offene Fragen und wartende Entwicklungen

Eine wissenschaftliche Arbeit kann nicht gut sein, wenn sie nur Antworten liefert. Eine wissenschaftliche Antwort ist wahrscheinlich genau dadurch gekennzeichnet, daß sie neue Fragen stellt. Und so ist es auch mit dieser Arbeit. Es sind einige Punkte offen geblieben, die es gilt, weiter zu lösen. Viele dieser Punkte führen das System in andere Richtungen, in die Grenzbereiche anderer Disziplinen der Informatik und auch darüber hinaus.

12.1.1 “Seperation of Duty”

Wie schon erwähnt, ist es an der Zeit, nun endlich das beschriebene Konzept der “seperation of duty” zu realisieren. Das Konzept ist beschrieben und die Fundamente für die Integration sind geschaffen. Was fehlt, ist eine Implementierung, die am PRZ zu diesem Zeitpunkt vorangetrieben wird.

12.1.2 Datenaustausch mit verschiedenen Key-Servern

Eines der wichtigsten Kritikpunkte des Systems war immer, daß SME-Teilnehmer nicht auch am Mailverkehr mit S/MIME-Teilnehmern beteiligt sein können. Es müßte an dieser Stelle also eine S/MIME-Implementierung gemacht und die Kommunikation mit diversen Keyservern realisiert werden. Dies gilt auch für Keyserver, die z.B. PGP-Schlüssel halten.

12.1.3 Identifikation nach außen

Das interessante am SME ist, daß es mit diesem Produkt möglich ist, im Namen einer Organisation zu sprechen und im Namen dieser Organisation auch zu unterschreiben. Das entspricht sehr viel mehr dem Verwaltungskonzept, als die Kommunikation mit einer einzelnen Person. Wenn ich für das PRZ ein Buch einkaufe, dann gehe ich nicht in ein Geschäft und unterschreibe nur mit meinem “guten” Namen. Ich bekomme einen Vordruck der TU-Berlin mit einem Stempel und einer Unterschrift, die mich autorisiert, im Namen der TU-Berlin einen Einkauf zu tätigen. Genau das kann auch das SME.

Aber nach dem heutigen Konzept kann es auch nur das! Was meiner Meinung nach diesem Konzept noch fehlt, ist die Möglichkeit, wahlweise auch als Projektmitglied eines Projektes, in seiner Rolle als Systemadministrator oder auch in seiner Rolle als Einzelperson eine Unterschrift zu leisten.

Wie soll das SME diese Rollen unterscheiden? Ist es für eine Mitarbeiterin oder einen Mitarbeiter möglich, solche Unterscheidungen leicht zu machen? Oder wird das System damit wieder so komplex, das es nicht mehr benutzbar ist? Wie soll eine Person von außen die Rollen auseinanderhalten?

An dieser Stelle wäre es meiner Meinung nach durchaus sinnvoll weiterzudenken.

12.1.4 Hierarchisches Domänen-Konzept

Zu verschiedenen Gelegenheiten habe ich immer wieder angedeutet, daß es möglich ist, verschiedene SMEs miteinander zu verbinden. Selbstverständlich läßt sich das einfach machen, indem man vom befreundeten SME das ACM übernimmt, vorausgesetzt diese ACMs überschneiden sich nicht in der Begriffsgebung (was sehr schwierig werden sollte) und vorausgesetzt, man bekommt auch alle öffentlichen Schlüssel dazu, die zu diesem Modell gehören. Dieses Konzept nenne ich das Verschmelzungskonzept.

Ansonsten gäbe es noch die Möglichkeit, nur die nach außen hin wichtigen Personen oder Rollen der anderen Organisation (Repräsentanten) in das Modell aufzunehmen und allen den SMS-Schlüssel der anderen Organisation zu geben. Beim anderen SMS würde eine solche Mail behandelt werden, als käme sie von einem SMG der eigenen SME (mit allen Vor- und Nachteilen, die das mit sich bringt). Dieses Konzept nenne ich das Repräsentantenkonzept.

Es wäre jedoch auch denkbar, so etwas, wie Schlüssel für Domänen einzuführen, so daß ein echtes hierachisches Modell dabei herauskommen würde. Das würde der heutigen Organisation der TU am ehesten entsprechen (z.B. TU-Schlüssel, CS-Schlüssel, PRZ-Schlüssel usw.).

12.1.5 Workflowkonzepte

E-Mails eignen sich hervorragend zur Übertragung von Workflow-Daten. Bei einigen Brainstorming-Sitzungen am PRZ erläuterte ich bereits meine Idee vom Workflow-Attachment.

Gehen wir davon aus, daß das ACM ein statisches Abbild der Organisation ist. Dann könnte ein System, das diese Organisation und ihre Rollen und Zugehörigkeiten kennt, auch hierüber arbeiten.

Viele Formulare in der Verwaltung sind heute darauf aufgebaut, das diese mit der Hauspost von einer Bearbeiterin zu einem anderen Bearbeiter gesendet werden und dabei jeweils durch Unterschrift ein Arbeitsschritt bestätigt wird und Daten komplettiert werden.

Meine Idee war nun, solche Formulare abzubilden und als Attachments an die Mail anzuhängen. Dies kann über die gewohnten Mechanismen sicher geschehen. Unterschriften können in Form von digitalen Signaturen geleistet werden.

Ein solches System zu installieren, sollte relativ leicht sein, da auf das MIME-Attachment-Konzept zurückgegriffen und ein solches Workflow-Agent-Programm als Anzeigemodul (engl. viewer) für diesen MIME-Typ angegeben werden kann.

Diverse Möglichkeiten stünden hiermit offen, die bislang mit der Papiervariante nicht existierten. Werden die Formulare zu einem zentralen Rechner, wie dem SMS immer wieder zurückgesendet, so kann von dort auch regelmäßig ein Statusreport erfolgen. Personen können an wichtige Formulare erinnert werden. Ein Formular kann an mehreren Orten gleichzeitig sein. Die Unterschriften können zusammengeführt werden. Und bestimmte Abläufe können so automatisiert werden, indem korrekte Unterschriften beispielsweise bestimmte Aktionen auslösen.

12.1.6 Gruppenmailboxen

Benutzt man E-Mails als Auftragsverwaltung, steht man vor dem Problem, daß in einem Team einer nicht weiß, was der andere gemacht hat, wenn in einer Mail die Aufforderung zu einer Handlung enthalten ist (z.B. “Liebe Abteilung VI a, bitte schicken Sie mir die Statistiken des letzten Quartals!”). Entweder denkt jeder, der andere hätte gehandelt und nichts passiert oder alle handeln und die Arbeit wird doppelt gemacht. Absprachen sind die einzige Lösung. Was aber, wenn aus verschiedenen organisatorischen Gründen Absprachen nicht möglich sind?

Es gibt das Modell des gemeinsamen WWW-Mailers, das wir im Projekt entworfen haben und das zugunsten des SME-Konzeptes zunächst zurückgestellt wurde. Auf einen solchen WWW-Mailer könnte gemeinsam zugegriffen werden. Eine Person könnte verschiedene Mailboxen besitzen, die für ihn in einer zusammengefaßt werden. Eine Statusänderung könnte zu jeder E-Mail angezeigt werden, so daß implizierte Absprachen möglich wären.

Aber vielleicht gibt es ja noch eine andere Möglichkeit, um eine E-Mail-Gruppen-Arbeit zu ermöglichen.

12.1.7 Konfigurierbarkeit des SME

Neben den technischen Mängeln des SME gibt es auch noch Mängel, die rein organisatorischer Art sind. So müßte es eigentlich möglich sein, das Verhalten des SME der Sicherheitspolitik der Organisation anzupassen.

Das SME sollte also noch durch Konfigurationen erweitert werden, die das Verhalten des Systems beeinflussen. Solche Punkte sind:

- Es sollte möglich sein, zu konfigurieren, ob Klartextmail aus dem SME verschickt werden können oder ob nur verschlüsselte Mails möglich sind.
- Das Verhalten auf fehlerhafte Mails sollte einstellbar sein. Es sollte die Auswahl geben zwischen: Fehler führt zur Vernichtung der Mail, Fehler wird gemeldet, die E-Mail aber ausgeliefert, alle fehlerhaften Mails werden dem Sicherheitsexperten zur Überprüfung gegeben.
- Das Verhalten auf Attachments sollte konfiguriert werden können. So könnte man aus sicherheitspolitischen Gründen Attachments ganz untersagen (meiner Meinung nach eine schlechte Idee), den Versand von aktiven Inhalten untersagen und damit Viren unterbinden, Attachments automatisch oder manuell prüfen lassen.

Im praktischen Einsatz wird diese Liste sicherlich noch länger. Dieser letzte Punkt ist so etwas wie eine Fleißarbeit. Die Konzepte existieren, die Umsetzung ist teilweise sehr zeitaufwendig.

12.2 Fazit

Das SME ist ein vielversprechendes System, das heute an vielen Stellen eingesetzt werden könnte. Es beinhaltet ein großes Potential an Erweiterungsmöglichkeiten, die noch ausgeschöpft werden könnten. Interessenten für einen Einsatz sind vorhanden.

KAPITEL 13

Kritische Betrachtung des Systems

13.1 Der Arbeitsplatz “Verwaltung”

Die Einführung neuer Software bedeutet immer auch Veränderung des Arbeitsplatzes. Ein Softwaresystem erhebt den Anspruch, ein Werkzeug zu sein. Allgemein sollte man den Anspruch an ein Werkzeug geltend machen, daß es die Arbeit erleichtert und die Arbeit effektiver, leichter oder einfach nur angenehmer macht.

Viel zu viele Computersysteme wurden auf Arbeitsplätzen eingesetzt, die diesen Ansprüchen nicht gerecht wurden. Statt die Arbeit in irgendeiner Form zu verbessern, machten diese Systeme die Arbeit stupider, verlängerten die Dienstwege oder führten durch Ausfälle des Systems zur völligen Arbeitsunfähigkeit.

Soll ein System, wie das SME in der Verwaltung eingesetzt werden, so muß geprüft werden, ob sich der Arbeitsplatz und somit die Arbeit selbst verbessert oder zumindest nicht verschlechtert. Wie in der Einleitung und der Analyse beschrieben, sehe ich viele Möglichkeiten, die durch den Einsatz von sicheren erweiterten E-Mails entstehen. Ich muß mir aber auch die Frage stellen: Wie verändere ich einen Arbeitsplatz, der mit meinem SME ausgestattet wird.

In der Arbeitswissenschaft spricht man von Humankriterien, die erfüllt sein sollten, um den Anforderungen eines “guten” Arbeitsplatzes gerecht zu werden. “Gut” bedeutet in diesem Zusammenhang: Ein humaner Arbeitsplatz, der auf die Dauer die Arbeitenden nicht schädigt und eine qualitativ hochwertige und effiziente Arbeit ermöglicht. Diese Kriterien lauten:

1. Entscheidungsspielraum: Die Person muß in der Lage sein, innerhalb von möglichst weit gesteckten Grenzen, Entscheidungen selbst zu treffen. Dafür müssen die entsprechenden Grundlagen existieren, die sinnvolle Entscheidungen ermöglichen.

2. Angemessener zeitlicher Spielraum: Der Person muß genügend Zeit zur Verfügung stehen, um ihre Arbeit qualitativ hochwertig zu bewältigen.

3. Durchschaubarkeit und Eigeninitiative: Die Person muß in die Lage versetzt werden, die Arbeit, die sie zu verrichten hat, zu verstehen und eigene Initiativen aus den erkannten Zusammenhängen zu entwickeln.

4. Frei von organisatorischen und arbeitstechnischen Hindernissen: Die Arbeit darf nicht durch die Arbeitsorganisation oder irgendwelche äußeren Umstände behindert werden.

5. Arbeitstätigkeiten müssen ausreichend Bewegung zulassen: Die Person darf nicht an einen Ort gebunden sein. Die Tätigkeit muß so ausgerichtet sein, daß sie Bewegung einplant.

6. Arbeitsaufgaben müssen konkrete Kontakte zulassen: Das heißt, daß die Person mit möglichst vielen Sinnen ihre Arbeit begreifen soll (Wer ist der Mensch hinter der Akte, die ich gerade bearbeite? Wie sieht das Gebäude aus, dessen Räume ich gerade einteile?).

7. Variationsmöglichkeiten: Es darf nicht den einen korrekten Weg geben, sondern es sollten möglichst mehrere Varianten zulässig sein.

8. Möglicher arbeitsbezogener Kontakt: Arbeitskolleginnen und Kollegen sollten sich auch von Angesicht zu Angesicht gegenüberstehen können.

Die Qualität, die ein Werkzeug, wie das SME in einem Arbeitsumfeld haben kann, ist nicht nur vom Werkzeug selbst, sondern vor allem von seinem Einsatz abhängig. Die Kunst besteht also darin, das richtige, funktionierende Werkzeug auf sinnvolle Weise einzusetzen.

Die Einführung des SME wäre zunächst einmal eine zusätzliche Alternative. Mitarbeiterinnen und Mitarbeiter könnten weiterhin Akten, Anweisungen und Mitteilungen über die Hauspost verschicken, Telefonate führen, persönlich vorbeigehen oder aber die neue E-Mail benutzen. Bedingung wäre nur, daß die E-Mails regelmäßig gelesen würden.

Völlig absurd wäre die Idee, daß die Umstellung von Papierpost auf elektronische Post das Resultat hätte, daß mehr Arbeit geschafft werden würde. Die Arbeitenden würden nicht schneller werden. Nur würden die Kommunikationswege kürzer. Die einzigen Einsparungen könnten an den Stellen entstehen, an denen Mitarbeiterinnen und Mitarbeiter nichts weiter zu tun hätten, als auf neue Post zu warten. Mir ist persönlich von solchen Fällen noch nichts zu Ohren gekommen. Alles, was das E-Mailsystem erreichen könnte, wären kürzere und schnellere Wege. Die Akte, die heute eine Dienststelle verläßt, ist, wenn sie nicht direkt überbracht wird, erst am nächsten Tag (wenn überhaupt) bei der nächsten Stelle. Muß eine Akte also 4 Stellen durchlaufen, ist sie fühestens in einer Woche zu bearbeiten. Über das SME wäre eine Bearbeitung innerhalb eines Tages zu schaffen, weil die Wege kürzer sind. Die Bearbeitungszeit verändert sich nicht.

Durch ihre Geschwindigkeit ist die E-Mail eine Alternative zum Telefon. Ein Telefonat kann an der richtigen Stelle das optimale Arbeitsmittel sein. Ist eine Person jedoch in die Bearbeitung eines komplexen Vorgangs vertieft, so kann ein Telefonat den gesamten Arbeitsfluß unterbrechen. Die Anfrage am Telefon sorgt in diesem Fall dafür, daß die betroffene Person sich danach erst wieder einarbeiten muß. Für die anrufende Person kann es genauso frustrierend werden, wenn der andere Anschluß ständig besetzt ist. Hier kann die E-Mail Nerven schonen und im besten Fall auch Arbeiten effektiver und qualitativ besser machen.

Wichtig ist, daß bei einer Einführung des SME darauf zu achten ist, daß die anderen Kommunikationswege an sinnvollen Stellen weiter benutzt werden. Die E-Mail darf nur eine Ergänzung sein.

Eine solche SME-Einführung könnte selbstverständlich auch helfen, Papier und damit Ressourcen zu sparen. Aber Vorsicht bei der Einführung des papierlosen Büros: Allein die Arbeitshaltung bei nur bildschirmbezogenen Arbeiten kann auf Dauer wieder einen hohen Krankenstand und damit Engpässe in der Bearbeitung von Akten hervorrufen. Der Weg zum Aktenschrank ist nicht nur die Gelegenheit für eine kurze Bewegung, sondern auch eine Gelegenheit für die Augen, sich zu regenerieren.

Wie immer gilt also hier auch der Grundsatz: Im richtigem Maß eingesetzt, kann dieses System allen Betroffenen etwas bringen.

13.2 Beispiele aus der Praxis

In der Verwaltung laufen verschiedene Rundschreiben umher. Die meisten erreichen aus irgendwelchen Gründen die Betroffenen viel zu spät. So wurde mir berichtet, daß beispielsweise Stellenausschreibungen oft Wochen nach der Ausschreibungsfrist bei interessierten Personen ankamen. Damit hatten diese gar keine Chance mehr, sich auf diese Ausschreibung zu melden. Hier wird auf Grund

organisatorischer Unzulänglichkeiten das Gleichheitsrecht mit Füßen getreten. Wer zufällig (oder vielleicht gar nicht so zufällig) weiter vorn in der Verteilerliste steht, hat die Chance auf diesen Arbeitsplatz. Eine Verteilung über E-Mail würde hier das Gleichheitsrecht wieder herstellen, vorausgesetzt, jede Angestellte und jeder Angestellte hätte Zugang zu einem Computer, an dem er E-Mails empfangen kann.

Ein Gegenbeispiel aus unseren Modellabteilungen. Im Rahmen des E2S-Projektes wurde eine Vorstufe des SME zur Vernetzung von zwei Abteilungen eingeführt. Grundsätzlich wurde die Einführung der E-Mails hier mit Wohlwollen aufgenommen und auch sinnvoll genutzt. Eine Mitarbeiterin ließ sich jedoch von dieser Abteilung versetzen, weil sie psychische Probleme damit hatte. Die Probleme dieser Frau sind vor ihrem speziellen Hintergrund verständlich. Als langjährige Mitarbeiterin in der TU ist diese Frau den Umgang mit den ihr vertrauten Werkzeugen gewohnt. Seit einiger Zeit steht auch ein Computer für ihre Arbeit zur Verfügung. Da dieser Mitarbeiterin jedoch der Zugang zu dieser Technik fehlte, hielt sie sich weiter an die herkömmlichen, manuellen Methoden. Die Einführung der E-Mail zum Austausch der Vorgänge in ihrer Abteilung zwingt sie jedoch nun dazu, auf die EDV einzugehen. Das verändert ihr Berufsbild. Ihr Unwissen um Abläufe und Funktionen macht ihr Angst; aus ihrer Sicht passieren (durch Fehlbedienungen oder auch Programmfehler) nicht abschätzbare Dinge. Sie muß Arbeit doppelt machen, die früher schnell erledigt war. Für diese Frau ist das Kriterium 3 und das Kriterium 4 verletzt. Die Frage stellt sich, ob dies ein Einzelfall ist oder ob viele Kolleginnen und Kollegen Probleme mit einer solchen Umstellung haben.

13.3 In wieviel Tagen wurde Rom erbaut?

Auf diese Frage lautet die richtige Antwort: Nicht an einem Tag. Ich könnte die Idee nicht unterstützen, daß an einem Tag X in der gesamten Verwaltung irgendeiner Organisation in der Größe der TU-Verwaltung gesagt werden würde: Ab heute benutzen wir alle das SME. Eine solche Einführung müßte schrittweise erfolgen.

An erster Stelle müßte ein Pilotprojekt mit einigen sinnvollen Kommunikationspartnern durchgeführt werden. Wir könnten hier in der TU z.B. die Abteilungen weiter vernetzen und z.B. erst einmal 4 oder 5 Abteilungen Arbeitsabläufe erarbeiten und testen lassen. Dabei ist wichtig, daß die Mitarbeiterinnen und Mitarbeiter selbst an der Gestaltung des Schriftverkehrs mit dem SME mitarbeiten. Sie können sich am besten vorstellen, an welchen Stellen es bislang “gehakt” hat und was verbessert werden kann.

Zudem muß auch die Zuverlässigkeit der Software getestet und natürlich ständig verbessert werden. Evtl. entstehen bei der aktiven Arbeit mit dem System noch weitere Anforderungen, die bislang nicht bedacht wurden.

In einer zweiten Phase könnten alle anderen Abteilungen ebenfalls vernetzt werden. Nachdem alte und neue Organisationen eine Zeit lang nebeneinander existiert haben und keine weiteren Schwierigkeiten mit dem System zu erwarten sind, könnte eine vollständige Einführung stattfinden.

13.4 Verfügbarkeit des Systems

Wie bereits mehrfach erwähnt, ist die Software zwar implementiert und auch prototypisch getestet. Für ein echtes Produkt sollte es jedoch noch eine Vielzahl von Tests durchlaufen und entsprechend verbessert werden.

Man sollte nicht vergessen: Ist das System eingeführt, müssen sich die Kolleginnen und Kollegen auch darauf verlassen können. Fällt ein solches System aus, kann dies selbst, wenn der alte Weg noch zur Verfügung steht, zu erheblichen Problemen führen, da die neuen Arbeitsschritte darauf ausgerichtet sind.

Es muß also eine ausreichende Vorsorge für die Verfügbarkeit des Dienstes getroffen werden.

13.5 Sicherheit des Systems

Bei aller Vorsicht und Diskussion über die Sicherheit sollte eine Abwägung stattfinden, wie hoch ein Risiko bei der Einführung eines solchen Systems ist. Es muß bedacht werden, was passiert, wenn es wirklich einem Angreifer gelingt, an die Inhalte der E-Mails zu gelangen oder Unterschriften zu fälschen.

Von absoluter Wichtigkeit ist die ständige Weiterentwicklung des Systems, um es auf dem heutigen Stand der Technik zu halten. Das heißt, alle heute bekannten Angriffe abwehren zu können.

Es muß ferner in Erwägung gezogen werden, daß bei Bekanntwerden größerer Sicherheitslücken, das System für so lange wieder eingestellt werden muß, bis die Mängel beseitigt sind.

13.6 Ein Schritt nach dem nächsten

Meine Empfehlung wäre, das SME in der Praxis tatsächlich zu testen, sobald es in ausreichendem Maße unter Laborbedingungen getestet und verbessert wurde.

Um wirklich sicher zu gehen, daß das System tatsächlich eine Verbesserung des Arbeitsumfeldes darstellt, könnte auf Arbeitsanalysemethoden zurückgegriffen werden, wie z.B. das KABA-Verfahren, daß hier an der TU entwickelt wurde [Dun93].

Nach der Einführung des SME müßte die Analyse wiederholt werden und könnte ein meßbares Ergebnis für die Qualität des Produktes für die Arbeit geben.

Das SME ist in meinen Augen eine längst fällige Entwicklung. Sie kann meiner Meinung nach durchaus helfen, die Arbeit im Verwaltungsbereich zu verbessern. Das System ist aber in keiner Hinsicht ein Wundermittel. Die zu erwartenden Verbesserungen sind nicht zu überschätzen, sollten jedoch den investierten Aufwand in jedem Fall wieder ausgleichen. Wenn damit Dienstwege verkürzt werden können, und der eine oder andere Baum am Leben bleibt, weil er nicht zu Papier verarbeitet wird, hat das System schon etwas vollbracht.

Bei all diesen Einschätzungen darf jedoch nicht vergessen werden: Ich bin kein Verwaltungsangestellter. Ich kenne die Arbeit der Verwalterinnen und Verwalter nur aus den kurzen Kontakten mit mir als “Kunde” und vom Hörensagen. Ich mache das SME als Angebot. Ich überlasse es jedoch anderen, zunächst einmal herauszufinden, ob es seine Erwartungen erfüllt.

Ich kann mich meiner Verantwortung als Informatiker nicht entziehen, wenn es darum geht, Arbeitsplätze umzugestalten. Aus meiner heutigen Sicht bin ich dabei, damit etwas zu verbessern. Trotzdem muß die Entwicklung an jeder Stelle immer auch mit kritischen Augen betrachtet werden. Wir können alle gespannt auf das Ergebnis sein.

ANHANG A

Quellenverzeichnis

Amm96	Ammeraal, Leen, algorithms and data structures in C++, John Wiley & Sons, Chichester, New York u.a. 1996
Amm97	Ammeraal, Leen, STL for C++ programmers, John Wiley & Sons, Chichester, New York u.a. 1997
Barth98	Bartholdt, Jörg, Konzeption und Implementierung einer Zugriffskontrolle innerhalb eines virtuellen privaten Netzwerks für private und öffentliche Verwaltungen, Diplomarbeit, Technische Universität Berlin, März 1998
Busch98	Buschmann, Frank et.al., Pattern-orientierte Software-Architektur, Addison-Wesley, Bonn, Reading, Massachusetts u.a. 1998
Camp98	Camphausen, Igmar et. al., Probleme beim PGP-Einsatz in Zertifizierungsstellen und deren Lösung durch PGP2.6.3in und Open PGP, 5. Workshop DFN-Cert, 1998
Cert97	DFN-Cert Sicherheitsbulletins, http://www.cert.dfn.de/infoserv/dsb/
Cost97	Costales, Bryan et.al., sendmail, Cambridge, Köln u.a. 1997
Crisp96	Crispin, M., Internet Message Access Protocol - Version 4rev1, RFC 2060, University of Washington, 1996
Dun93	Dunckel, H. et. al., Kontrastive Aufgabenanalyse im Büro. Der KABA-Leitfaden. Zürich: vdf und Stuttgart: Teubner 1993
E2.4-97	Bartholdt, Jörg et. al., Deliverable E2.4: Secure Telecooperation Software, END-TO-END Security over the internet, 1997

F1-97	Barthildt, Jörg et. al., Deliverable F1: Secure Telecooperation Software, END-TO-END Security over the internet, 1997
Freed96	Freed. N et.al., Multipurpose Internet Mail Extensions (MIME), RFC 2045 - 2049, 1996
Gam96	Gamma, Erich, Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley, Bonn 1996
Heg94	Hegemann et. al., Datenschlösser / Grundlagen der Kryptographie, c't 1994, Heft 8, S. 230ff.
Herf97	Herfert, Michael, Security Enhanced Mailing Lists, IEEE Network and Internet Security Vol. 11 No. 3, p. 30-33, May/June 1997
Holz98	Holz, Helmut et.al., LINUX für Internet und Intranet, Thomson Publishing Company, Bonn, Albany u.a. 1998
Kos98	Kossakowski, Klaus-Peter, Virtuelle Private Netzwerke - Einsatzmöglichkeiten und Techniken, 5. Workshop DFN-Cert, Hamburg 1998
Krol95	Krol, Ed, Die Welt des Internet - 1. Aufl., O'Reilly/International Thomson Verlag, Bonn 1995
Leff90	Leffler, Samuel J. et. al., The Design and implementation of the 4.3 BDS UNIX operating system, Addison-Wesley Publishing Company, Reading, Massachusetts u.a. 1990
Linn93	Linn, J. et. al., Privacy Enhacement for Internet Electronic Mail, RFC 1421 - 1424, 1993
Myers96	Myers, J. et. al., Post Office Protocol - Version 3, RFC 1939, 1996
Nag96	Nagel, Klaus et. al., User Requirements for Administration Sectors, E2S/WP.C/TUB/001, Berlin February 1996
Petz96	Petzold, Charles, Windows 95 Programmierung, Microsoft Press, Unterschleißheim 1996
Rud98	Christen, Rudolf et. al., Snif+ von Unix nach NT portiert, iX Magazin für Professionelle Informationstechnik, S. 86-90, Juli 1998
Schn96	Schneier, Bruce, Angewandte Kryptographie, Protokolle, Algorithmen und Sourcecode in C, Addison-Wesley Publishing Company, Bonn, Reading, Massachusetts u.a. 1996
Schön92	Schöning, Uwe, Theoretische Informatik kurz gefaßt, BI-Wiss.-Verlag, Mannheim, Leipzig u.a. 1992
SigG	Gesetz zur digitalen Signatur, http://www.iid.de/rahmen/iukdgbt.html

SigV	Verordnung zur digitalen Signatur, http://www.iid.de/rahmen/sigv.html
Secm95	SecureMail User's Guide for the Compartmented Mode Workstation (CMW), Secure War, July 1995
Sim97	Seimet, Uwe, Unter Kontrolle, Quelltextverwaltung mit GNU CVS, iX September 1997
Strou92	Stroustrup, Bjarne, Die C++ Programmiersprache, Addison Wesley, Bonn, Paris u.a. 1992
Tack96	Tacket, Jack et.al., LINUX das Kompendium, Einführung - Arbeitsbuch - Nachschlagewerk, Markt und Technik Buch- und Software-Verlag, München 1996
Tich79	Tichy, Walter, Software Development Control Based on Module Interconnection, Proc. 4th International Conference on Software Engeneering, Munich 1979
Tisch95	Tischer, Michael et. al., PC intern 5, Data Becker, Düsseldorf 1995
Visio97	Developing Visio Solutions, Visio International Limited, Ireland 1997
Will97	Willms, Gerhard, C++ Das Grundlagenbuch, Data Becker, Düsseldorf 1997
X.400	X.400, Message Handling Services: Message handling system and service overview, ITU-T X-Series Recommendations, Rev. 1, Apr 1993
X.500	X.500, Information Technology - Open systems Interconnection - The Directory: Overview of concepts, models and services, ITU-T X-Series Recommendations, Rev. 1, Dec 1994
X.509	X.509, Information Technology - Open Systems Interconnection - The Directory: Authentification framework, ITU-T X-Series Recommendations, Rev. 1, Dec 1994
Zim90	Zimmermann, Phillip, PGP Pretty Good Privacy, Deutsche Übersetzung, Abel Deuring und Christopher Creuzig, FoeBud e.V. Bielefeld, 1990-1994

ANHANG B

Glossar

Access Control Model	Modell zur Abbildung einer Organisationsstruktur samt ihrer Personen und Rollen.
ACM	Abk. für Access Control Model
ACMI	Interpreter für ACMs. Kann Methoden auf Rollen dieser Modelle ausführen.
aout	Definiertes Format für den Aufbau von ausführbaren Dateien bei UNIX - Derivaten.
ARPA	Abk. für Advanced Research Projects Agency: Abteilung des Verteidigungsministeriums der USA. Das ARPA-Netz ist ein Netzerk für experimentelle und theoretische Untersuchungen auf dem Gebiet der Rechnernetze.
ARPA-Message	Elektronische Nachricht im definierten Format.
ASCII	Abk. für American Standard Code for Information Interchange: 7-Bit-Kodierung für die Darstellung von Ziffern, Buchstaben und Sonderzeichen.
BBS	Abk. für Bulletin Board System: Rechner, die mit einer speziellen Software und einem Modem ausgestattet sind und zum Austausch von Nachrichten und von Software über Modem dienen. In der Regel nicht (oder nur asynchron) vernetzte Systeme.
Bit	Kleinste Informationseinheit in der EDV. Trägt den Zustand 1 oder 0. 8 Bit ergeben ein Byte, welches als Kombination ausreicht, um ein Zeichen zu repräsentieren oder einen Wert zwischen 0 und 255.
Bitfolge	Eine Menge von Bits mit einer definierten Reihenfolge.
Bitstrom	siehe Bitfolge
Body	Der Nachrichtenbestandteil einer E-Mail im Gegensatz zum Header.
Daemon	Serverprozeß ohne Benutzungsschnittstelle. Stellt einen Dienst zur Verfügung und führt in der Regel Logfiles.
Dämon	siehe Daemon
Client	Programm, daß einen Dienst über Netzwerk in Anspruch nimmt (z.B. Mailclient dient zum Senden und Empfangen von E-Mails benutzt dabei aber entsprechende Server, um die Aufgabe durchzuführen).
CMW	Abk. für Compartmented Mode Workstation: Definition der Defense Intelligence Agency (DIA) zu einem sicheren Betriebssystem, das sensible Daten in einem Compartment halten kann, auf das von restlichen System nicht zugegriffen werden kann.
E2S	Abk. für das EU - Projekt END-TO-END SECURITY over the Internet.
elf	Alternatives Format zu aout.
E-Mail	Abk. für Electronic Mail, elektronische Post.
ESMTP	Abk. für Enhanced SMTP (siehe SMTP)

Fingerprint	Fingerprints werden benutzt, um die Echtheit eines Schlüssels zu prüfen. Es handelt sich um eine relativ kurze Zahlenfolge, die man als Mensch leicht vergleichen kann. Es ist schwer einen zweiten Schlüssel zu erzeugen, der genau den selben Fingerprint hat, wie ein anderer.
Firewall	Konzept zur Sicherung von Teilnetzen in größeren Netzen. Bezeichnet einen Rechner oder dessen Software, die zwischen das zu schützende und das ungeschützte Netz geschaltet wird, um Wächterfunktionen durchzuführen.
Gateway	Programm oder Software auf einem Programm, welche ein Netz mit einem anderen verbindet oder sogar verschiedene Protokolle ineinander übersetzt.
GMD	Die GMD ist das nationale Forschungszentrum für Informationstechnik. Gesellschafter der GMD ist die Bundesrepublik Deutschland.
GNATS	Datenbank zur Erfassung von Fehlern bei Software. Unterliegt der GNU General Public Licence. Arbeit auf Basis von E-Mails zur Fehlerübermittlung.
GNU	Abk. für GNU is not UNIX: Projekt mit der Zielsetzung: Entwicklung eines völlig freien UNIX-Systems. Für die rechtliche Grundlage der freien Verbreitung der Software sorgt die GNU General Public Licence (GNU-GPL).
Hashfunktion	Einwegfunktion, die als Prüfsumme benutzt werden kann. Es ist schwer, einen zweiten Bitstrom zu erstellen, der eine identische Prüfsumme besitzt.
Hashwert	Ergebnis einer Hashfunktion bei gegebenem Bitstrom.
Header	Steuerinformationen zur Zustellung einer E-Mail. Hat die Aufgaben eines Briefumschlags. Enthält Informationen, wie Absender, Empfänger, Datum, usw.
Host	Rechner auf dem Serverprogramme laufen, der also Dienste zur Verfügung stellt.
IDEA	Abk. für International Data Encryption Algorithm: Verschlüsslungsverfahren, das von Xuejia Lai und James Massey entwickelt wurde.
IP	Abk. für Internet Protokoll.
IP-Nummer	Eindeutige Identifikation eines Rechners im Internet.
IP-Packet	Das Internet Protokoll ist ein paketvermitteltes Netzwerk. Das heißt, daß die Informationen in Pakete verpackt werden und über das Netz geschickt werden. Diese Pakete haben ein wohldefiniertes Format.
ITU	Abk. für International Telecommunication Union: Standartisierungsgremium.
Java	Objektorientierte Programmiersprache, die es dank einer sog. Java Virtual Machine zuläßt, ein Programm auf verschiedenen Computerplattformen laufen zu lassen.
Klasse	Ein abstrakter Datentyp mit Attributen und Methoden. Also ein Typ, mit den dazugehörigen Funktionen und Datenrepräsentationen.
LAN	Abk. für Local Area Network. Beschreibt ein Netwerk aus Computern mit geringer räumlicher Distanz (z.B. eine Abteilung einer Firma kann durch ein LAN vernetzt sein).
LDAP	Abk. für Lightweight Directory Access Protocol: Zugangsprotokoll für X.500 Verzeichnisdienst.
Logbuch	In diesem Zusammenhang ein Protokoll mit einer Liste von Ereignissen, die von Programmen angelegt wird. Dient der Kontrolle oder auch zur Fehlersuche.
Logdatei	Datei, in der das Logbuch geführt wird.
Logfile	engl. Bezeichnung für Logdatei
Mailbody	siehe Body
Mailheader	siehe Header
Mailingliste	Verteiler für E-Mails als Diskussionsforum. Man kann sich in einer Mailingliste eintragen lassen, um an den Diskussionen teilzunehmen oder einfach nur Informationen zu bestimmten Themen zu erhalten. Es gibt Mailinglisten zu den unterschiedlichsten Themen. Manche sind nur zum Lesen, manche zum Lesen und Schreiben.
Mailserver	Programm zum Versenden von E-Mails oder der Rechner auf dem ein solches Programm läuft. In unserem Zusammenhang SMTP - Server insbesondere das Programm Sendmail als spezielle Implementierung.

Methode	Eine Operation auf einen speziellen Datentypen. Methoden sind Funktionen, die zu einer bestimmten Klasse gehören (z.B. Klasse Complex als Implementierung der komplexen Zahlen mit der Methode add() zur Adition).
MIME	
Objekt	Eine Instanz einer Klasse. Ein spezielles Individuum eines komplexen Typs (z.B. Klasse “Fahrrad” mit der Instanz “das Fahrrad von Thomas Hildmann”).
Objektmodell	Modell zur Darstellung der beteiligten Klassen mit Relationen (also Beziehungen), die zwischen den Klassen herrschen.
Objektinteraktionsgraph	Darstellung der Interaktionen (wie z.B. Methodenaufrufe) zwischen Objekten der Klassen. Beantwortet die Frage: Welches Objekt sendet welche Nachrichten an welches andere Objekt.
Operationsgraph	Stellt die wichtigsten Eigenschaften einer Operation (hier etwa gleich zu setzen mit Methode oder Funktion) zusammen, z.B. welche Attribute verändert werden, welche Nebeneffekte diese Opereation hat usw.
Packetfilter	Eine Art einer Firewall, die jedes IP-Paket analysiert und anhand von Regeln (woher kommt dieses Paket, wohin geht es und welchen Dienst nutzt es) weiterleitet, zurückweist oder ignoriert.
Parser	Zerlegt eine gegebene künstliche Sprache in Objekte, die dann von anderen Softwarekomponenten interpretiert werden können.
PD	Abk. für Public Domain: Beschreibt kostenlose Produkte. Siehe auch GNU.
PEM	Abk. für Privacy-Enhanced Mail: Standard zur Verschlüsslung von E-Mails.
PGP	Abk. für Pretty Good Pravacy: Von Phillip Zimmermann entworfenes Programm zur Verschlüsselung mit eigenem Datenformat. Hat sich zum Quasistandard in der Verschlüsselung erhoben.
Pipe	Datenkanal, welcher vom Betriebssystem zur Kommunikation der Prozesse untereinander zur Verfügung gestellt wird.
POP3	
Public Key	Der öffentliche Teil einer asymmetrischen Verschlüsselung. Dient der Verschlüsselung einer Nachricht für einen bestimmten Empfänger oder der Überprüfung einer digitalen Signatur.
Private Key	Der private Teil einer asymmetrischen Verschlüsselung. Dient der Entschlüsselung einer Nachricht oder der digitalen Signierung.
Proxy	Dient als Stellvertreter für ein anderes Programm. Übernimmt das Protokollverhalten in zwei Richtungen so, daß die beteiligten Komponenten nicht vom Stellvertreterobjekt beeinflußt werden.
Router	Ein Computersystem, das Daten zwischen zwei Netzwerken transferiert, die dasselbe Protokoll verwenden. Die physikalischen Gegebenheiten der Komponenten können unterschiedlich sein; ein Router kann z.B. Daten zwischen einem Ethernet und einer Standleitung transferieren.
Secret Key	siehe Private Key
Server	Programm das einen Dienst über Internet zur Verfügung stellt. Oft wird auch der Rechner als Server bezeichnet, auf dem ein oder mehrere Serverprogramme laufen.
Session Key	Ein symmetrischer Schlüssel, der in der Regel zufällig bestimmt wird und für einen Übertragungsabschnitt gültig ist und danach verworfen wird.
Smartcard	Meist Plastikkarte in der Größe einer Checkkarte, die einen Chip enthält. Dieser Chip enthält einen Prozessor, ein Programm und einen Datenspeicher für die Schlüssel. Daten können an die Smartcard übertragen werden, die dann die Verschlüsselung oder Signierung übernimmt. Analog läuft auch die Entschlüsselung und Prüfung. Es ist nicht möglich von außen an den privaten Schlüssel zu gelangen.
SME	Abk. Secure Mail Environment: Beschreibt das gesammte Konzept, wie im Kapitel „Automatisierung von sicherem elektronischen Schriftverkehr“ auf Seite 25 beschrieben wird.

SMG	Abk. Secure Mail Gateway: Programm, das die automatische Ver- und Entschlüsselung von E-Mails an der Schnittstelle zum LAN übernimmt.
SMS	Abk. Secure Mail Server: Zentraler Punkt im SME, welcher E-Mails entgegennimmt, die Auflösung von logischen E-Mailadressen durchführt und für die Empfänger entsprechend verschlüsselt.
SMTP	
Socket	Modell zur Interprozeßkommunikation, beschreibt den Endpunkt einer Kommunikation. Ein weit verbreiteter Mechanismus zur Kommunikation zwischen Client- und Server-Prozessen.
SPT	Abk. Secure Proxy Tools: Ein SMTP - Proxy und ein POP3 - Proxy, der zwischen die Kommunikation von Mailclient und SMTP- bzw. POP3-Server geschaltet wird und die automatische Ver- und Entschlüsselung von E-Mails übernimmt.
Task	
TCP/IP	Abk. für Transmission Control Protocol: Protokoll, das auf dem Internet Protokoll aufsetzt.
Thread	
Timeline Diagramm	Diagramm, daß mit Hilfe von sog. Szenarien (Ablaufbeispielen) die Abläufe der Kommunikation zwischen den Objekten der untersuchten Klassen und der Außenwelt (bezeichnet als System, auch Agenten genannt) darstellt.
UUCP	Abk. für Unix-to-Unix-Copy: Älteres Protokoll zum Austausch von Dateien unter UNIX-Rechnern auch über Entfernungen hinweg (z.B. über Modem, später auch über das Internet). Wurde lange zur für die Übertragung von E-Mails benutzt.
VPN	Abk. für Virtual Private Network: Ein Netzwerk in einem Netzwerk. Basiert darauf, daß eine kryptographisch gesicherte Verbindung über das unsichere Netz aufgebaut wird und ein neues “virtuelles” Netzwerk über den gesicherten Kanal geleitet wird.
WAN	Abk. für Wide Area Network: Steht im Gegensatz zum LAN. Beschreibt Netzwerkverbindungen mit hoher räumlicher Ausdehnung. Es stehen verschiedenste Techniken hierfür zur Verfügung (z.B. Kupferdraht, Glasfaser, Funkstrecke, Satelitenfunk).
WINSPT	Abk. für Windows Secure Proxy Tools: SPT-Variant (siehe SPT), die für Windows 95/98 und Windows NT geschrieben wurde.
XOR	Logische Funktion exklusives Oder. Definiert als 0 = 1 xor 1 = 0 xor 0 1 = 1 xor 0 = 0 xor 1

ANHANG C

Datenlexikon

TABELLE 1. Datenlexikon

Name	Art	Beschreibung
Mail	Klasse	Repräsentation einer E-Mail. Eine solche Mail besteht aus genau einem Header und einem Body. Das System baut auf Filteroperationen auf einer E-Mail auf.
Header	Klasse	Die Interne Abbildung eines E-Mailheaders nach RFC 822. Die für die Weiterverarbeitung wichtigsten Header-Zeilen werden in Attributen der Klasse gespeichert, der gesamte restliche Header wird im Attribut other gespeichert.
Body	Klasse	Diese Klasse nimmt den Inhalt einer E-Mail auf, also den eigentlichen Text, der über die Mail verschickt wird.
Acmi	Klasse	Diese Klasse kapselte die Kommunikation mit dem Access Control Model Interpreter. Methodenaufrufe dieser Klasse werden über das TCACM - Protokoll an einen ACMI übertragen. Antworten werden ausgewertet zurückgeliefert.
Pkcrypt	Klasse	Modell eines Public-Key-Algorithmus mit Standardoperationen. Diese Klasse steht stellvertretend für verschiedene PK-Algorithmen. Die jeweiligen Algorithmen werden mittels Strategie-Methode ausgewählt.
Empfängerliste	Klasse	Eine Liste von Empfängern einer E-Mail.
Empfänger	Klasse	Ein Eintrag aus der Empfängerliste. Speichert einen E-Mail-Empfänger in den Attributen user, host und remark.
Cryptedmail	Klasse	Wurde später eingeführt. Ist ein Spezialfall einer Mail, auf die Verschlüsselungs- und Signierungsmethoden angewendet werden können.
Cryptaddr	Klasse	Diese Klasse besitzt nur einen Konstruktur und zwei Methoden zum Auslesen der Attribute. Sie dient zum Zerlegen einer Zeile des ACMI in die Bestandteile ARPA-Adresse und X400-Adresse (entspricht dem Schlüssel-Identifizierer).
Sendmail	Agent	Das Programm Sendmail ist ein spezieller SMTP-Server, der eine weite Verbreitung im Internet besitzt und mittels “Mailer”-Konzept externe Programme aufruft, um diesen E-Mails zur Weiterverarbeitung zu übergeben. Sendmail ist in den Diagrammen gleich zu setzen mit SMTP-Server.

TABELLE 1. Datenlexikon

Name	Art	Beschreibung
SMTP - Server	Agent	Siehe Sendmail.
löst_auf	Relation	Beschreibt den Vorgang der Auswertung einer logischen E-Mail-adresse des AC-Modells in ein oder mehrere Paare von ARPA-Adressen und X400-Adressen.
signiert_mit	Relation	Signaturen werden üblicherweise durch Verschlüsselung mit dem privaten Teil eines Public - Key - Paares realisiert. Der Empfänger kann durch Entschlüsselung mit dem öffentlichen Teil des Schlüssels schlußfolgern, ob der Text manipuliert wurde.
wird_verschlüsselt_mit_für	Relation	Ein E-Mailtext wird üblicherweise mit dem öffentlichen Schlüssel eines Empfängers verschlüsselt. Zu lesen ist die Relation: Body wird verschlüsselt mit PublicKey für Empfänger.
headersigniert_mit	Relation	Beim SME wird auch der E-Mailheader signiert. Dies geschieht mit dem privaten Schlüssel des Absenders.
Empfängt	Relation	Der Agent Sendmail empfängt E-Mails, die er an das SME zur Weiterbearbeitung weiterreicht.
Sendet	Relation	Der Agent Sendmail kann zum Versenden von E-Mails benutzt werden.
maildaten_lesen	Systemoperation	Der Agent Sendmail stößt das einlesen der E-Maildaten an. Das heißt, daß Sendmail selbst die Daten an das System weitergibt.
headersignatur_prüfen	Systemoperation	Die SMG oder SMS Steuereinheit löst das Überprüfen der Headersignatur aus.
bodysignatur_prüfen	Systemoperation	Die SMG oder SMS Steuereinheit löst das Überprüfen der Bodysignatur aus.
body_entschlüsseln	Systemoperation	Die SMG oder SMS Steuereinheit löst das Entschlüsseln des E-Mailtextes aus.
signiere_header	Systemoperation	Die SMG oder SMS Steuereinheit löst das Signieren des Headers aus.
verschlüssle_body	Systemoperation	Die SMG oder SMS Steuereinheit löst das Verschlüsseln des Headers aus.
signiere_body	Systemoperation	Die SMG oder SMS Steuereinheit löst das Signieren des Mailtextes aus.
versenden	Systemoperation	Von einer Steuereinheit wird das Versenden der E-Mail mittels SMTP ausgelößt.
Attribute und Operationen der Klasse Pkcrypt		
me	Attribut	Beinhaltet den eigenen Schlüsselidentifizierer (Bezeichnung des eigenen privaten Schlüssels).
pin	Attribut	PIN, Paßwort bzw. Passphrase mit der auf den privaten Schlüssel zugegriffen werden kann.
pkpath	Attribut	Verzeichnispfad, Paar aus Rechner und Port oder Gerätepfad für den Zugriff auf private und öffentliche Schlüssel (z.B. Software-PSE, Smartcard oder Keyserver).
pgppath	Attribut	Pfad zum ausführbaren Programm PGP.
pempath	Attribut	Wie pgppath nur für PEM.
pgppassenv	Attribut	Die Passphrase von PGP kann außer über Tastatur nur über eine Environmentvariable übergeben werden. Damit bei Programmstart der alte Variableninhalt jedoch nicht verloren geht, wird die Variable vorher gespeichert, damit ggf. noch einmal darauf zurückgegriffen werden kann.
pgppassset	Attribut	Nimmt den Wert TRUE an, wenn vor dem Programmstart die PGP Environmentvariable gesetzt war und jetzt überschrieben wurde.

TABELLE 1. Datenlexikon

Name	Art	Beschreibung
pktype	Attribut	Gibt an, welcher Algorithmus ausgewählt wurde (z.B. PGP, PEM, KH, ...).
print	Operation	Gibt den Inhalt aller Attribute aus.
sign	Operation	Signiert den String " in" und liefert das Ergebnis in "out" zurück.
mkxsig	Operation	Baut eine Signatur des Headers auf, welche in den Mailheader eingesetzt werden kann.
unmkxsig	Operation	Macht die Wirkung von mkxsig rückgängig.
crypt	Operation	Verschlüsselt die Zeichenkette in mit dem öffentlichen Schlüssel von to und liefert das Ergebnis in out.
readPin	Operation	Ließt die PIN von der Standardeingabe ein.
decrypt	Operation	Entschlüsselt die Zeichenkette "in" und liefert das Ergebnis in "out".
check	Operation	Liefert TRUE, wenn die Signatur der Eingabezeichenkette "in" korrekt war und gibt über "out" den signierten Text ohne Signatur aus.
Attribute und Operationen der Klasse Mail		
mailcmd	Attribut	Shellkommando zum Starten des externen Programms zum Versenden von E-Mails (z.B. "/usr/bin/sendmail").
h	Attribut	Mailheader: Stellt eine Art Briefumschlag mit Informationen über Absender, Empfänger etc. dar.
b	Attribut	Mailbody: Beinhaltet den E-Mailtext.
forwardMail	Operation	Sendet die Mail mittels externem Mailprogramm (mailcmd) an den angegebenen Empfänger.
returnToSender	Operation	Schickt die Mail zurück an den Absender.
Attribute und Operationen der Klasse Cryptedmail		
checkHeaderSignature	Operation	Überprüft die Signatur des E-Mailheaders. Intern wird ein Pkcrypt.unmkxsig() und ein Pkcrypt.check() aufgerufen. Das Ergebnis wird als TRUE im Falle einer intakten Signatur zurückgeliefert, sonst FALSE.
checkBodySignature	Operation	Überprüft die Signatur des E-Mailtextes. Ermittlung der Integrität der Signatur über den Mailbody erfolgt durch Aufruf von Pkcrypt.check(). Es wird dann TRUE im Falle einer intakten Signatur zurückgeliefert, sonst FALSE.
decryptBody	Operation	Entschlüsselt den Mailtext. Es wird ein Infotext hinzugefügt, der über das Entschlüsseln aufklärt.
signHeader	Operation	Unterschreibt wie beschrieben den Header der E-Mail mittels Pkcrypt.mkxsig() und Pkcrypt.sign().
cryptBody	Operation	Verschlüsselt den E-Mailbody für den angegebenen Empfänger.
signBody	Operation	Unterzeichnet den E-Mailbody.
Attribute und Operationen der Klasse AddressList		
addresses	Attribut	Die Liste der Adressen.
p	Attribut	Positionzeiger (zeigt auf den jeweils aktuellen Datensatz)
add	Operation	Fügt eine Adresse hinzu.
isempty	Operation	Liefert TRUE, wenn keine Adressen in der Liste stehen.
size	Operation	Liefert die Zahl der enthaltenen Adressen zurück.
first	Operation	Liefert die erste Adresse aus der Liste.
next	Operation	Liefert die jeweils nächste Adresse aus der Liste.

TABELLE 1. Datenlexikon

Name	Art	Beschreibung
StringToAddressList	Operation	Wandelt einen String, der eine Liste von Adressen (im RFC 822 - Format) enthält in eine Adresslist um.
print	Operation	Gibt den Inhalt der Adressliste RFC 822 konform aus.
Attribute und Operationen der Klasse Address		
username	Operation	Der Anteil Benutzername in einer E-Mailadresse im ARPA-Adressformat (z.B. remark <username@hostname>).
hostname	Operation	Der Anteil Rechnername mit Domain einer ARPA-Mailadresse.
remark	Operation	Meist der bürgerliche Name (engl. realname) des Benutzers.
setRemark	Operation	Ändert das Attribut remark.
setAddress	Operation	Füllt die Attribute username, hostname und remark aus einem gegebenen String, der eine syntaktisch korrekte ARPA-Mailadresse enthält. Verschiedene Formate sind möglich.
isempty	Operation	TRUE, wenn keine Attribute gesetzt sind.
Attribute und Operationen der Klasse Acmi		
serverhost	Attribut	Rechner auf dem das ACMI läuft.
requeststring	Attribut	Komplette Anfrage an das ACMI im TCACM - Format.
data	Attribut	Antwort des ACMI im TCACM - Format.
port	Attribut	Der Port, auf dem der ACMI arbeitet.
sockd	Attribut	Der Socketdescriptor, also der Zeiger auf die Datei, die die TCP/IP Socket zum ACMI repräsentiert.
request	Operation	Ein komplett frei definierbarer Request an das ACMI. Liefert Antwort als String.
resolveAddr	Operation	Anfrage “resolveAddr” an das ACMI (siehe TCACM - Protokoll). Liefert die aufgelösten E-Mailadressen und Schlüsselidentifizierer. In jeder Zeile ein Paar, ARPA-Adresse und X400-Adresse durch Komma getrennt.
Attribute und Operationen der Klasse Body		
body	Attribut	Ein String, der den gesamten Mailbody hält.
clear	Operation	Löscht den Mailbody. body enthält danach den leeren String.
addline	Operation	Fügt Zeile hinten an body an.
insertline	Operation	Fügt Zeile vorn in body ein.
toString	Operation	Liefert den String body zurück.
Attribute und Operationen der Klasse Header		
to	Attribut	Empfänger der Mail. Kann eine einzelne ARPA-Mailadresse sein oder eine Liste von Empfängern. Attribut speichert Zeile als einzelnen String (kann bei Bedarf mittels Addresslist.stringToAddressList() zerlegt werden).
from	Attribut	Mailadresse des Absenders.
cc	Attribut	Enthält keinen, einen oder mehere Mailempfänger, die eine Kopie der E-Mail bekommen sollen (siehe to).
subject	Attribut	Betreff der E-Mail. Wird als String abgelegt.
date	Attribut	Absendedatum der E-Mail. Wird meist vom SMTP-Server des Absenders oder vom Mailclient angegeben.
sig	Attribut	Enthält gegebenenfalls die Signatur über den Mailheader (“X-Signature:”). Sonst ist sig leer.
other	Attribut	Enthält den restlichen Header (die Zeilen, für die keine Attribute zur Verfügung stehen).

TABELLE 1. Datenlexikon

Name	Art	Beschreibung
Attribute und Operationen der Klasse Cryptaddr		
arpaaddr	Attribut	Adresse im ARPA-Format (z.B.: hildi@cs.tu-berlin.de)
x400addr	Attribut	Adresse im X400-Format (z.B.: C=de,O=TU-Berlin,CN=Thomas.Hildmann)

TABELLE 2. Übersetzung der Bezeichnungen

deutsche Bezeichnung	englische Bezeichnung oder C++ Umsetzung
signieren	sign
prüfen	check
verschlüsseln	crypt
erzeuge_x-signatur	mkxsig
extrahiere_x-signatur	unmkxsig
maildaten_lesen	operation<<
versenden	forwardMail
signiere_header	signHeader
headersignatur_pruefen	checkHeaderSignature
verschlüssle_body	cryptBody
signiere_body	signBody
bodysignatur_prüfen	checkBodySignature
body_entschlüsseln	decryptBody
Empfängerliste	AddressList
Empfänger	Address

ANHANG D

Bekannte Fehler (GNATS-Report)

Im Folgenden die bekannten Fehler aus der GNATS Datenbank. Aktualisierte Fassungen können unter der WWW-Adresse: http://lexx.mash-gmbh.com/cgi-bin/wwwgnats.pl abgefragt werden.

```
     CATEGORY   RELEASE SEVE$ R$ STATE     CLASS    ORIGINAT$ SYNOPSIS
5    tub-sme    alpha $ Crit  h$ closed    sw-bug   Spresa_D$ Makefiles in SMS und SMG fehlen
6    tub-sme    alpha $ Noncr h$ closed    sw-bug   Thomas_H$ Leerzeilen in Mails vermehren sich
7    tub-sme    Alpha $ Ser   p$ closed    sw-bug   Thomas_H$ Signaturcheck nach Pkcryptpgp::unmkx-
sig() schlaegt fehl
8    tub-sme    Releas$ Ser   h$ closed    sw-bug   Thomas_H$ Allgemeine Schutzverletzung bei Mails
ohne Body
9    tub-sme    Releas$ Noncr h$ open      sw-bug   Thomas_H$ Absturtz bei fehlerhaftem PEM - Schlüs-
sel
13   tub-sme    Alpha $ Crit  h$ analyzed  sw-bug   Thomas H$ Headersignatur umfaßt nicht den Mail-
body
14   tub-sme    Alpha $ Noncr h$ analyzed  sw-bug   Thomas H$ Mailadressen werden im Header nicht
normalisiert
15   tub-sme    Alpha $ Noncr h$ open     sw-bug   Thomas H$ Große Mails können mit dem WINSPT nicht
übertragen werden
16   tub-sme    Alpha $ Ser   h$ open      sw-bug   Thomas H$ Keine Kontrolle bei WINSPT ob Verbin-
dung von lokalem Rechner kommt
17   tub-sme    Alpha $ Noncr h$ open     change-$ Thomas H$ Gesprächigkeit von WINSPT konfigurier-
bar machen
18   tub-sme    Alpha $ Noncr h$ open     support  Thomas H$ Fehler beim Prüfen der Headersignatur
19   tub-sme    Alpha $ Ser   h$ open      sw-bug   Thomas H$ Absender innerhalb des SME fälschbar
```

```
>Number:         6
>Category:       tub-sme
>Synopsis:       Leerzeilen in Mails vermehren sich
>Confidential:   no
>Severity:       non-critical
>Priority:       medium
>Responsible:    hildi
>State:          closed
>Class:          sw-bug
>Submitter-Id:   hildi
>Arrival-Date:   Wed Sep 02 15:21:00 MEST 1998
>Last-Modified:  Sun Nov 08 17:18:48 MET 1998
>Originator:     Thomas_Hildmann
>Organization:
hildi
>Release:        alpha 1.1
>Environment:
Lexx, Aventuria Linux
>Description:
Zwischen Header und Body der Mail kommwn irgendwo im SMS oder SMG immer mehr
```

```
Leerzeilen dazu.
>How-To-Repeat:
Mail ueber SMGSEND verschicken
>Fix:

>Audit-Trail:
State-Changed-From-To: open-closed
State-Changed-By:      Thomas Hildmann <hildi@mash-gmbh.com>
State-Changed-When:    Sun Nov 08 17:16:54 (MEZ) Mitteleuropäische Zeit 1998
State-Changed-Why:     Konnte nicht mehr festgestellt werden
>Unformatted:
```

```
>Number:         7
>Category:       tub-sme
>Synopsis:       Signaturcheck nach Pkcryptpgp::unmkxsig() schlaegt fehl
>Confidential:   no
>Severity:       serious
>Priority:       medium
>Responsible:    pristina
>State:          closed
>Class:          sw-bug
>Submitter-Id:   hildi
>Arrival-Date:   Tue Sep 08 23:13:01 MEST 1998
>Last-Modified:  Sun Nov 08 17:19:23 MET 1998
>Originator:     Thomas_Hildmann
>Organization:
>Release:        Alpha 1.1
>Environment:
Linux 2.x
>Description:
Das cryptedmailtest zeigt, wie eine Signatur mit PGP im Header erstellt und
wieder geprueft wird. Beim Pruefen wird jedoch ein “false” als Antwort
geschickt.

Entweder liegen die Probleme im mkxsig() oder im unmkxsig() des Pkcryptpgp.
>How-To-Repeat:
cryptedmail starten und gueltigen PGP Schluessel angeben.
%0
>Fix:

>Audit-Trail:
Responsible-Changed-From-To: hildi-pristina
Responsible-Changed-By: Thomas_Hildmann
Responsible-Changed-When:
Responsible-Changed-Why:
Spresa hat die Routinen geschrieben

State-Changed-From-To: open-closed
State-Changed-By:      Thomas Hildmann <hildi@mash-gmbh.com>
State-Changed-When:    Sun Nov 08 17:17:31 (MEZ) Mitteleuropäische Zeit 1998
State-Changed-Why:     Fehler ist korrigiert.
>Unformatted:
```

```
>Number:         8
>Category:       tub-sme
>Synopsis:       Allgemeine Schutzverletzung bei Mails ohne Body
>Confidential:   no
>Severity:       serious
>Priority:       medium
>Responsible:    hildi
```

```
>State:          closed
>Class:          sw-bug
>Submitter-Id:   hildi
>Arrival-Date:   Mon Oct 12 08:12:01 MEST 1998
>Last-Modified:  Sun Nov 08 17:18:15 MET 1998
>Originator:     Thomas_Hildmann
>Organization:
hildi
>Release:        Release 1.0
>Environment:
Windows 95, Microsoft Mail, P200MMX, 64MB
>Description:
Wird eine Mail übertragen, die gar keinen Body besitzt,
stürzt das Programm ab. Man muß erst an der SW vorbei
die leere Mail löschen.
>How-To-Repeat:
Mail nur mit Subject erzeugen
>Fix:

>Audit-Trail:
State-Changed-From-To: open-closed
State-Changed-By:      Thomas Hildmann <hildi@mash-gmbh.com>
State-Changed-When:    Sun Nov 08 17:16:21 (MEZ) Mitteleuropäische Zeit 1998
State-Changed-Why:     Mit leeren Mails gibt es keine Probleme mehr.
>Unformatted:
```

```
>Number:         9
>Category:       tub-sme
>Synopsis:       Absturtz bei fehlerhaftem PEM - Schlüssel
>Confidential:   no
>Severity:       non-critical
>Priority:       medium
>Responsible:    hildi
>State:          open
>Class:          sw-bug
>Submitter-Id:   hildi
>Arrival-Date:   Mon Oct 12 08:15:01 MEST 1998
>Last-Modified:
>Originator:     Thomas_Hildmann
>Organization:
hildi
>Release:        Release 1.0
>Environment:
Windows 95, P200MMX, 64MB
>Description:
Fehlt der Pfad zur PSE oder ist PIN bzw. Identity falsch oder fehlt
die ID des SMS, die zum verschlüsseln benötigt wird, stürzt das
Programm ab.

Abhilfe schafft das korrekte setzen dieser Parameter. Es sollten
entsprechende Überprüfungsroutinen eingebaut werden.
>How-To-Repeat:
PSE umbenennen oder Parameter im Programm ändern.
%0
>Fix:
>Audit-Trail:
>Unformatted:
```

```
>Number:         13
```

```
>Category:       tub-sme
>Synopsis:       Headersignatur umfaßt nicht den Mailbody
>Confidential:   no
>Severity:       critical
>Priority:       high
>Responsible:    hildi
>State:          analyzed
>Class:          sw-bug
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:36:01 MET 1998
>Last-Modified:  Sun Nov 08 21:57:24 MET 1998
>Originator:     Thomas Hildmann
>Organization:
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel
>Description:
Zur Zeit wird in der Implementierung nur der Mailheader signiert und bei
der Signierung nicht der Body einbezogen.
Damit ist es möglich einen falschen Body unter einen Header zu hängen.
>How-To-Repeat:
Gültigen Header erzeugen, der an den Angreifer adressiert ist.
Dann beliebigen Mailbody an diesen Header hängen und erneut absenden.
Das SMS wird die Mail für den Angreifer verschlüsseln.
>Fix:
Fix müßte in der Methode mkxsig() gemacht werden. Der Body muß hier einfach
noch
dazukommen.
>Audit-Trail:

State-Changed-From-To: open-analyzed
State-Changed-By: Thomas_Hildmann
State-Changed-When:
State-Changed-Why:
Fix ist bekannt, muß jedoch noch getestet werden.
>Unformatted:
```

```
>Number:         14
>Category:       tub-sme
>Synopsis:       Mailadressen werden im Header nicht normalisiert
>Confidential:   no
>Severity:       non-critical
>Priority:       low
>Responsible:    hildi
>State:          analyzed
>Class:          sw-bug
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:36:02 MET 1998
>Last-Modified:  Sun Nov 08 21:58:41 MET 1998
>Originator:     Thomas Hildmann
>Organization:
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Bei der Headernormalisierung werden die Routinen zur Normalisierung einzel-
ner
Adressen nicht verwendet. Die Implementierung ist komplett in der Klasse
Address. Zur Erzeugung ganzer Listen könnte Addresslist benutzt werden.
```

>How-To-Repeat:
Das Fehlen dieser Normalisierung hat bislang noch zu keinem Fehler geführt
könnte aber, wenn auf dem Weg z.B. aus “Thomas Hildmann” <hildi@in-brb.de>
die Adresse hildi@in-brb.de (Thomas Hildmann) gemacht werden würde.
Beide Adressen sind korrekt!

>Fix:

>Audit-Trail:
State-Changed-From-To: open-analyzed
State-Changed-By: Thomas_Hildmann
State-Changed-When:
State-Changed-Why:
Entsprechende Methoden sind schon implementiert, müssen
jedoch noch integriert werden.
>Unformatted:

>Number: 15
>Category: tub-sme
>Synopsis: Große Mails können mit dem WINSPT nicht übertragen werden
>Confidential: no
>Severity: non-critical
>Priority: medium
>Responsible: hildi
>State: open
>Class: sw-bug
>Submitter-Id: hildmann
>Arrival-Date: Sun Nov 08 21:36:03 MET 1998
>Last-Modified:
>Originator: Thomas Hildmann
>Organization:
MASH
>Release: Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Bei der Übertragung größerer E-Mails über das WINSPT entsteht im WINSPT
ein Deadlock.

>How-To-Repeat:
E-Mail mit größerem Attachment versenden (z.B. Bilddaten).

>Fix:

>Audit-Trail:
>Unformatted:

>Number: 16
>Category: tub-sme
>Synopsis: Keine Kontrolle bei WINSPT ob Verbindung von lokalem Rech-
ner kommt
>Confidential: no
>Severity: serious
>Priority: high
>Responsible: hildi
>State: open

```
>Class:          sw-bug
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:39:01 MET 1998
>Last-Modified:
>Originator:     Thomas Hildmann
>Organization:
MASH
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Es wird zur Zeit noch nicht überprüft, ob die Verbindung des Mailclient
wirklich vom lokalem Rechner kommt.

>How-To-Repeat:
Verbindung von einem anderen Mailclient über einen fremden PC aufbauen.
WINSTP wird an stelle des PC-Eigentümers unterzeichnen.
Genauso funktioniert auch eine POP3 Abfrage. Nur ist hier auch noch ein
Paßwort nötig, daß zuvor aus dem Netz gesnifft werden muß.

>Fix:


>Audit-Trail:
>Unformatted:
```

```
>Number:         17
>Category:       tub-sme
>Synopsis:       Gesprächigkeit von WINSPT konfigurierbar machen
>Confidential:   no
>Severity:       non-critical
>Priority:       low
>Responsible:    hildi
>State:          open
>Class:          change-request
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:41:01 MET 1998
>Last-Modified:
>Originator:     Thomas Hildmann
>Organization:
MASH
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Es wäre schön. wenn man einstellen könnte, ob WINSPT bei jeder Mail eine
Meldung bzw. eine Frage bringt oder ob WINSPT schweigend arbeitet.

>How-To-Repeat:
Es handelt sich um keinen Fehler sondern nur um einen Korrekturvorschlag.

>Fix:


>Audit-Trail:
>Unformatted:
```

```
>Number:         18
```

```
>Category:       tub-sme
>Synopsis:       Fehler beim Prüfen der Headersignatur
>Confidential:   no
>Severity:       non-critical
>Priority:       low
>Responsible:    hildi
>State:          open
>Class:          support
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:44:01 MET 1998
>Last-Modified:
>Originator:     Thomas Hildmann
>Organization:
MASH
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Wird für die FROM: - Zeile ein falscher Wert im Mailcient konfiguriert,
nimmt Sendmail nach der Signatur eine Korrektur vor. Diese führt dann
beim Empfänger zu Fehlermeldungen.

Es ist darauf zu achten, daß die FROM: - Zeile entsprechend den Angaben
des Mailservers eingestellt sind.

>How-To-Repeat:
Falsche Eintragung bei der Mailadresse im Mailclient machen und Mail über
WINSPT verschicken.

>Fix:


>Audit-Trail:
>Unformatted:
```

```
>Number:         19
>Category:       tub-sme
>Synopsis:       Absender innerhalb des SME fälschbar
>Confidential:   no
>Severity:       serious
>Priority:       high
>Responsible:    hildi
>State:          open
>Class:          sw-bug
>Submitter-Id:   hildmann
>Arrival-Date:   Sun Nov 08 21:48:01 MET 1998
>Last-Modified:
>Originator:     Thomas Hildmann
>Organization:
MASH
>Release:        Alpha 1.2
>Environment:
System: Windows 95 schlepp 4.10 intel

>Description:
Beim Entwurf der Klassen wurde vergessen, daß das SMS prüfen muß, ob die
Signatur mit dem Absender übereinstimmt. Ob also in der FROM - Zeile die
E-Mailadresse steht, die zur Signatur paßt.
Kann ermittelt werden über resolveAddr, wird zur Zeit aber nicht gemacht.
```

Klasse Pkcrypt müßte entsprechend erweitert werden.
Ferner müßten entsprechende Änderungen im SMS vorgenommen werden.

>How-To-Repeat:
Mail mit gefälschter FROM - Zeile innerhalb des SME verschicken.

>Fix:

>Audit-Trail:
>Unformatted: