# Großer Beleg

**zum Thema**

# Eine *capability*-basierte Lösung zur Unterbindung von Spam

**Frank Herrmann**

**Technische Universität Dresden**
**Fakultät Informatik**
**Institut für Systemarchitektur**
**Professur Betriebssysteme**

**25.Oktober 2002**

Verantwortlicher Hochschullehrer:
Prof. Dr. Hermann Härtig

# Inhaltsverzeichnis

**1 Einleitung** **1**

**2 Grundlagen** **3**

2.1 Arten von Spam . . . 3
2.1.1 Werbung . . . 3
2.1.2 Kettenbriefe . . . 3
2.1.3 Hoaxes . . . 3
2.2 Email . . . 3
2.2.1 Aufbau von Emailadressen . . . 3
2.2.2 Aufbau einer Email . . . 4
2.2.3 Kontrollfluss vom Absender zum Empfänger . . . 4
2.3 Java Apache Mail Enterprise Server . . . 5
2.3.1 Eigenschaften . . . 6
2.3.2 Matcher und Mailets . . . 6
2.3.3 Konfiguration . . . 7
2.3.4 Mail, MailAddress, MailetContext . . . 8
2.4 MySQL . . . 8
2.5 HMAC-MD5 . . . 8
2.6 Bestehende Lösungen gegen UCE . . . 9
2.6.1 Filter auf Basis regulärer Ausdrücke . . . 9
2.6.2 Sperrung von Rechnern/Domainen . . . 9
2.6.3 Bayes-Filter . . . 9
2.6.4 Wegwerfadressdienste . . . 10
2.6.5 Berichterstattungsdienste, Adressverfolgungsdienste . . . 10
2.6.6 Teergruben . . . 10
2.6.7 Absenderverifikation mit SMTP . . . 11
2.6.8 Tagged Message Delivery Agent (TMDA) . . . 11
2.6.9 Vergleich aller Verfahren . . . 12
2.7 Zyklenopfer . . . 13

**3 Entwurf** **14**

3.1 Verfahren . . . 14
3.1.1 Erster Schritt . . . 14
3.1.2 Verbesserung der Skalierbarkeit . . . 15
3.1.3 Geplanter zweiter Schritt . . . 16
3.2 Ansatzpunkt der Filterung . . . 16
3.3 Capability-Sicherung . . . 17
3.4 Datenhaltung . . . 17

**4 Implementierung** **18**
4.1 Vorbemerkung . . . 18
4.2 Filterung . . . 18
4.3 Automatischer Antwortmechanismus . . . 18
4.4 Capability-Sicherung . . . 20
4.5 Datenhaltung . . . 20
4.6 Konfiguration . . . 22
4.7 Administration . . . 22
4.8 Beispielszenario . . . 23
4.8.1 Ausgangssituation . . . 23
4.8.2 Registrierung . . . 23
4.8.3 Kommunikation . . . 23
4.8.4 Administration . . . 24

**5 Bewertung** **25**
5.1 Einsatz . . . 25
5.2 Praktische Tests . . . 25

**6 Zusammenfassung und Ausblicke** **27**

**A Glossar** **28**

**B Literaturverzeichnis** **29**

# 1 Einleitung

Der Name SPAM (Akronym für *SPiced HAM*[1]) ist von der "Hormel Foods Corporation", einer Wurstfabrik, in über 100 Ländern der Welt markenrechtlich geschützt. Völlig vergeblich, denn mittlerweile verbindet kaum noch jemand Spam mit Würzfleisch. In der Umgangssprache des Internet verwendet man das Wort Spam für aufdringliche elektronische Postwurfsendungen. Die offizielle Bezeichnung dafür lautet *Unsolicited Commercial Email* (UCE, unerwünschte Werbemail). Dass ausgerechnet Würzfleisch als Metapher für ebenso überflüssige wie störende Kommunikation herhalten muss, geht der Legende nach auf einen Klassiker der englischen Komikertruppe Monthy Python zurück. Der Sketch aus dem Jahr 1970 spielt in einem Restaurant, das ausschließlich Speisen mit Würzfleisch anbietet. Ein Gast bittet flehentlich um eine Mahlzeit ohne Spam, doch seine Bitte wird von lautem Gesang eines Wikinger-Chores "Spam, herrlicher Spam, wundervoller Spam!" erstickt.

Genau wie in diesem Sketch ist der Benutzer eines elektronischen Postsystems einem UCE-Versender hilflos ausgeliefert. Emailadressen werden in großem Stil aus dem Internet herausgefiltert und an werbefreudige Firmen weiterverkauft. Laut dem Marktforschungsinstitut IDC transportiert das Internet täglich fast 30 Milliarden Emails. Davon sind etwa 6 Milliarden von den Empfängern weder angefordert noch erwünscht, also UCE.

UCE ist deshalb so verbreitet, weil es für den Versender sehr billig ist, tausende von Emails innerhalb kürzester Zeit zu versenden. Im Gegensatz dazu ist der Erhalt und die Bearbeitung der Nachrichten für den Empfänger teuer. Spam ist aber von den Empfängern höchst unerwünscht. Daher muss diese Relation ins Gegenteil verkehrt, also der Versand erheblich erschwert werden! Ein weiterer Aspekt eines Abwehrsystems ist die Benutzerfreundlichkeit. Nur einfach einzusetzende, komfortabel bedienbare und damit auch für einen Nicht-Experten nutzbare Lösungen können einen hohen Verbreitungsgrad erreichen. Wenn alle Benutzer des Internet Kryptographie-Experten wären, könnte man beispielsweise nur noch authentifizierte Mail erlauben. Die Authentifikation mit Hilfe von *Pretty Good Privacy* (PGP, Verschlüsselungssoftware) wäre eine denkbare Möglichkeit. Ein durchschnittlicher Internetnutzer wird aber einen solchen Aufwand nicht in Kauf nehmen wollen. Insbesondere das Verstehen der Grundlagen der Kryptographie und die Installation der entsprechenden Werkzeuge stellt eine zu große Hürde dar.

Stattdessen benötigt man einen schrittweisen Einstieg für den Nutzer. Ein erster Schritt darf keine umfangreichen Kenntnisse erfordern, muss aber trotzdem effektiv gegen Spam sein. Ähnliches gilt für Provider und Administratoren. Ein einfaches und wirkungsvolles System muss relativ schnell einsetzbar und später erweiterbar sein. Nachträgliche Erweiterungen setzen umfangreiches Wissen voraus, verbessern aber das System in hohem Maße. Im vorliegenden Beleg wird die Idee einer *capability*-basierten Lösung zur Unterbindung von Spam vorgestellt.

[1] engl. für Würzfleisch

## Gliederung

Im folgenden Kapitel werden die wichtigsten Arten von UCE beschrieben, Aufbau und Funktionsweise von Email erklärt, ein auf JAVA basierender Mailserver vorgestellt sowie bereits vorhandene Lösungen untersucht. In Kapitel 3 wird der Entwurf, insbesondere das *capability*-basierte Verfahren, erklärt. Zusätzlich werden Vorüberlegungen hinsichtlich Filterungspunkt, Datenhaltung, Konfiguration und Administration dargelegt. Kapitel 4 behandelt die Implementation. Vorgestellt wird die programmtechnische Umsetzung aller Aspekte des Entwurfes. In Kapitel 5 erfolgt eine Bewertung der praktischen Realisierung und eine Einschätzung der erreichten Ergebnisse. Den Abschluss der Arbeit bildet Kapitel 6. Es beinhaltet eine Zusammenfassung und zeigt mögliche Ansatzpunkte für weiterführende Arbeiten auf.

Nachfolgend einige Hinweise zu den Typographie-Konventionen in dieser Arbeit:

Fachwörter sind beim ersten Auftreten im Text kursiv gedruckt. In Klammern folgen Abkürzung und deutsche Erklärung. Ein Beispiel ist *Application Programming Interface* (API, Programmierschnittstelle). Programmanweisungen und Programmausschnitte sind in Schreibmaschinenschrift gedruckt, beipielsweise die Methode `getSubscriberState(String recipientEmailAddress)`. Zustände, Parameter, Argumente oder Konstanten sind mit KAPITÄLCHEN hervorgehoben.

# 2 Grundlagen

## 2.1 Arten von Spam

UCE kann in vielfältigen Ausprägungen bei dem Empfänger eintreffen. Als Motivation für eine neue Methode gegen Spam sollen in diesem Kapitel die wichtigsten Arten betrachtet werden.

### 2.1.1 Werbung

"Verdienen Sie 3.000 Euro in einer Woche!", "Sie haben gewonnen!" oder "Lassen Sie sich diese Chance nicht entgehen!". Solche Nachrichten beinhalten meist Angebote, die, beim Wort genommen, eigentlich gar nicht ausgeschlagen werden können: "Finanzielle Unabhängigkeit", "10.000 Euro in 30 Tagen" oder sogar eine "Umkehrung des Alterungsprozesses" wird versprochen. Ganz zu schweigen von wenig glaubhaften Zeugnissen virtueller Großzügigkeit wie geschenkten Firmenanteilen oder zinslosen Bankkrediten. Alle diese Angebote dienen letztendlich nicht dem Empfänger sondern dem finanziellen Interesse des Versenders.

### 2.1.2 Kettenbriefe

Kettenbriefe breiten sich über viele Jahre hinweg im Netz aus. Die Empfänger werden darin meist angehalten, die Email an möglichst viele Adressaten weiterzuleiten. Dabei wird behauptet, dass jede versandte Email gezählt werden würde. Im Nachrichtentext wird nicht nur an Habgier ("Bill Gates zahlt für jede Email 50 Euro..."), sondern auch an Gutmütigkeit ("Erfüllen Sie einem todkranken Kind einen letzten Wunsch...") und Besorgnis ("Wenn Sie diese Mail nicht weitersenden, wird Ihnen etwas Furchtbares geschehen...") appelliert. Letztendlich handelt es sich immer um einen fadenscheinigen Vorwand für eine völlig überflüssige Rundsendung.

### 2.1.3 Hoaxes

*Hoaxes* (Scherz, Finte) sind gefälschte Virenwarnungen, welche an eine große Anzahl von Adressaten versendet werden. Ziel ist es, einen massiven Verbreitungsgrad zu erreichen, um Mailsysteme zu belasten. Der eigentliche Virus ist die Email selbst. Durch eine im Idealfall exponentielle Verbreitung legt sie Mailserver lahm oder überflutet die Postkästen von Nutzern. Eine umfangreiche Auflistung von Hoaxes ist in [01] zu finden.

## 2.2 Email

### 2.2.1 Aufbau von Emailadressen

Emailadressen eignen sich sehr gut für die Unterbringung der *capability*. Adressen haben die allgemeine Form:

| user | @ | <subdomain.> | domain | . | top−level−domain |
|---|---|---|---|---|---|
| Bsp.: fh15 | @ | inf. | tu−dresden | . | de |

Der Teil *user* (Nutzerkennung) ist entweder ein eingetragener Nutzer auf der Empfängermaschine oder ein entsprechend gesetzter *alias* (Pseudonym). Danach folgen (eventuell mehrere) optionale *subdomains* (Unterdomänen) und eine *domain* (Domäne), die den Rechnernamen und die Organisation benennen. Der letzte Teil ist die *top-level-domain* (oberste Domäne).

### 2.2.2 Aufbau einer Email

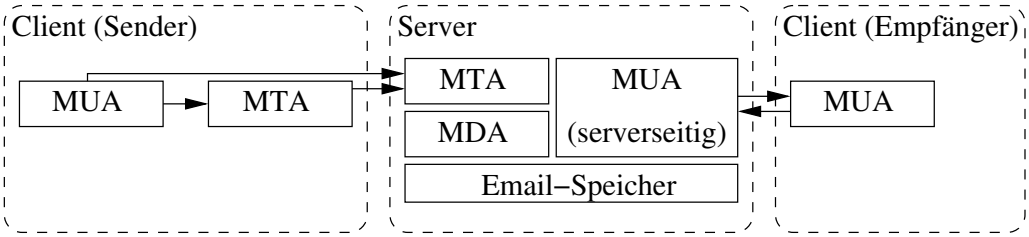
Der aktuelle Standard für den Aufbau einer Email ist in *request for comments 822* (RFC822, Internet Standard 822) [02] beschrieben. Danach besteht eine Email aus 3 wesentlichen Teilen: *envelope* (Umschlag), *header* (Kopf) und *body* (Rumpf).

- Der Umschlag enthält in Analogie zu einem Postbrief die Absender- und die Zieladresse. Der Umschlag wird benötigt, damit eine Email versendet werden kann.
- Der Kopf enthält Informationen für den Empfänger, z.B. zurückgelegter Weg, Absender-, Antwort- und Empfängeradresse, Betreff, Sendedatum und ID der Email. Es können im Kopf außerdem weitere anwendungsspezifische Zusatzinformationen gesetzt werden.
- Der Rumpf enthält den eigentlichen Inhalt der Nachricht. Dieser kann im einfachsten Fall reiner ASCII-Text sein, durch *Multipurpose Internet Mail Extensions* (MIME, universale Email-Erweiterung) können zusätzliche Zeichensätze gesetzt und dadurch Sonderzeichen verwendet werden. Mit Hilfe von MIME wird auch ein Anhängen beliebiger Dateien an die Nachricht ermöglicht.

### 2.2.3 Kontrollfluss vom Absender zum Empfänger

Das *Message Transfer System* (MTS, Nachrichtentransportsystem) wird durch drei Programmgruppen gesteuert: *Mail User Agent* (MUA, Email-Benutzerprogramm), *Mail Transport Agent* (MTA, Transportsystem) und *Mail Delivery Agent* (MDA, Zustellungssystem). Ein MUA, auch Mail-Klient genannt, wird vom Nutzer verwendet, um Emails zu versenden bzw. zu empfangen. Die Hauptaufgaben des MUA bestehen in der Formatierung der Eingabe, der Ergänzung von Metainformationen und dem Empfangen und Absenden der Email per spezifiziertem Empfangs- bzw. Versandprotokoll. Hauptsächlich verwendet man heute als Empfangsprotokoll das *Post Office Protocol 3* (POP3, Empfangsprotokoll Version 3) oder das *Interactive Mail Access Protocol* (IMAP, interaktives Zugriffsprotokoll). MUA's implementieren ausserdem die Ablage der empfangenen Emails in einem standardisierten

Format. Ein Beispiel für ein solches Format ist das *mbox*-Format, welches alle Nachrichten in einer Datei speichert. Im Gegensatz dazu verwendet das *mh*-Format für jede Nachricht eine separate Datei. Beispiele für MUA's sind Microsoft Outlook oder Netscape Messenger. Zur Zeit ist das *Simple Mail Transfer Protocol* (SMTP, einfaches Email Versandprotokoll) Standard als Versandprotokoll von Email. Die Nachricht wird an den MTA gesandt. Dieser leitet mit Hilfe von anderen MTA's sowie dem *Domain Name System* (DNS, verteilter Namensdienst) die Nachricht an die Zielmaschine weiter.




Dabei wird ein *Mail Exchange record* (MX record, Namenseintrag für Email) benötigt, welcher vom DNS bereitgestellt wird. Der MTA verwendet diesen, um den Namen der Zielmaschine herauszufinden. Ein Cache speichert ermittelte Informationen, um nachfolgende Nachrichten, welche diese Domäne als Ziel haben, schneller verarbeiten zu können. Oft werden sogar mehrere Maschinen für die Email-Bearbeitung einer Domäne definiert und erhalten einen entsprechenden DNS-Eintrag. Diese Redundanz soll Server- und Leitungsausfälle umgehen und einen sicheren Email-Verkehr ermöglichen. Beispiele für MTA's sind die MTA-Softwarepakete sendmail[03], qmail[04] und exim[05]. Falls die Zielmaschine nicht erreichbar ist oder die Nachricht nicht sofort weitergeleitet werden kann, wird letztere in eine Warteschlange eingefügt und später bearbeitet. Beim Erreichen der Zielmaschine definiert der MTA, welcher Nutzer die Nachricht erhalten soll, und übergibt sie an den MDA. Wenn die Zielmaschine nach einer gewissen Zeit immer noch nicht erreicht werden kann, wird die Nachricht an den Absender zurückgeschickt. Dies wird als *bouncing* (abprallen) bezeichnet. Der MDA kann die Nachricht im primären Email-Speicher ablegen oder Filteroperationen durchführen und die Nachrichten in verschiedenen Sekundärspeichern verteilen. Von dort können sie vom MUA des Empfängers gelesen und nutzerfreundlich dargestellt werden. Beispiel eines verbreiteten MDA ist die Software procmail[06].

## 2.3 Java Apache Mail Enterprise Server

Apache James wurde als Basis für die praktische Realisierung des *capability*-basierten Verfahrens ausgewählt. Daher soll er in den folgenden Abschnitten näher betrachtet werden.

### 2.3.1 Eigenschaften

Ein Server, der sowohl als MDA als auch MTA agiert wird auch als Mailserver bezeichnet. Der *Java Apache Mail Enterprise Server* (James, Apache Mailserver)[07] ist ein portabler Mailserver, welcher derzeit die Protokolle SMTP, POP3 und *Net News Transport Protocol* (NNTP, Transportprotokoll für Diskussionsforen) beherrscht. Unterstützung von IMAP ist in naher Zukunft geplant. Er ist vollständig in Java programmiert und unterliegt der Apache Software Lizenz[08]. Apache James besitzt folgende Merkmale:

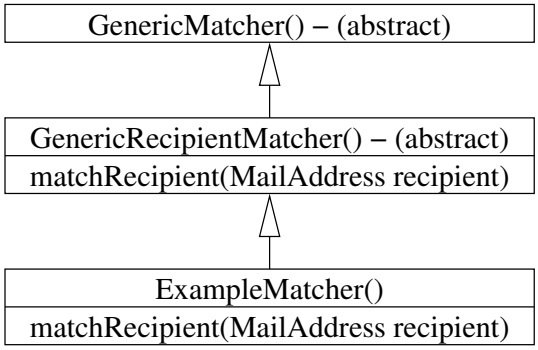
- **Portabilität** – Apache James ist eine komplett in Java 2/JavaMail 1.2[09] entwickelte Applikation und damit auf praktisch jedem entsprechend javafähigen System lauffähig.
- **Protokollabstraktion** – Im Gegensatz zu vielen anderen Mailservern werden Protokolle als eine Art "Kommunikationssprache" behandelt, welche die Regeln zur Kommunikation zwischen Server und Klienten bestimmt. Damit ist Apache James nicht an ein bestimmtes Protokoll gebunden und jederzeit erweiterbar.
- **Ressourcenabstraktion** – Der Ressourcenzugriff erfolgt abstrakt über definierte Schnittstellen. Der Server ist in großem Maße modular aufgebaut.
- **Vollständigkeit** – Apache James benötigt keine weiteren Lösungen oder Dienste.
- **Mailet-Unterstützung** – Apache James unterstützt das *Apache Mailet Application Programming Interface* (Apache Mailet API, Apache Mailet Programmierschnittstelle). Ein Mailet implementiert die Logik, mit der eine Email serverintern verarbeitet wird. Diese Logik kann in die Verarbeitungskette des Mailservers integriert werden. Mailets ermöglichen somit eine vollständige Einflussnahme auf den Weg einer Email durch den Server. Beispiele ihrer Anwendung sind Email-zu-Fax-Konverter, Email-zu-SMS-Konverter oder Sprachübersetzer.
- **Sicheres und paralleles Design** – Basierend auf der Apache *JServ Servlet Engine* (JSSE, Java Servletumgebung)[11] hat Apache James ein sicherheitsorientiertes Konzept. Außerdem ist der Server *multi-threaded* (interne parallele Verarbeitung) realisiert, um Leistung und Skalierbarkeit zu ermöglichen.

### 2.3.2 Matcher und Mailets

Die Mailet-API, beschrieben in [10], baut auf der JavaMail-API auf und benutzt einen ähnlichen Ansatz wie die Spezifikationen für den Umgang mit Servlet-Methoden und den entsprechenden Objekthierarchien. Sie umfasst zwei Kernklassen: Matcher und Mailet. Beide sind unter objektorientierter Sicht *interfaces* (Schnittstellen) mit den abstrakten Implementationen `GenericMatcher` und `GenericMailet`. Ein Matcher entscheidet darüber, ob

eine Nachricht im Mailserver verarbeitet werden soll, während ein Mailet die Nachricht verarbeitet. Dabei übernimmt der Mailserver die Instantiierung und Verwaltung der Objekte.

Der Matcher ist also sozusagen für die Wegeleitung innerhalb des Servers zuständig. Dazu existiert eine abstrakte Klasse `GenericRecipientMatcher`, die speziell Empfängeradressen auswertet. Streng genommen bekommt der Matcher eine ganze Sammlung von Objekten übergeben. `GenericRecipientMatcher` iteriert über diese Sammlung und erzeugt daraus ein einzelnes Objekt. Dadurch muss nur noch die `matchRecipient(MailAddress recipient)` Methode überschrieben werden. Folgendes UML-Diagramm demonstriert die Erstellung des Matchers EXAMPLEMATCHER.




Das Mailet definiert Methoden zur Initialisierung, Behandlung und Zerstörung des Mailobjektes. Der Server ruft diese Methoden während eines Mailet-Lebenszyklus in folgender Reihenfolge auf:

1. Das Mailet wird erzeugt und dann mit der `init(MailetConfig config)` Methode initialisiert.
2. Alle Nachrichten, welche die `service(Mail mail)` Methode aufrufen, werden entsprechend behandelt.
3. Das Mailet wird aus dem Pfad entfernt und mit der `destroy()` Methode zerstört. Schliesslich wird der *Java Garbage Collector* (automatische JAVA Speicherverwaltung) aktiv und gibt den belegten Speicherplatz frei.

### 2.3.3 Konfiguration

`ServletMailetConfig` erlaubt dem Server, beliebige Initialisierungsparameter an das Mailet zu übergeben. Bei James sind diese Parameter in der Datei `config.xml` festgelegt:

```
<mailet match="testeKriterium" class="sendeMail">
<prioritaet>wichtig</prioritaet>
</mailet>
```

In obigem Beispiel ruft der Server bei jeder Mail den Matcher TESTEKRITERIUM auf. Bei positiver Rückmeldung wird das Mailet SENDEMAIL aufgerufen. Dabei wird zusätzlich der Parameter PRIORITAET mit dem Argument WICHTIG übergeben.

### 2.3.4 Mail, MailAddress, MailetContext

Weitere nützliche Mailet-API-Klassen sind `Mail`, `MailAddress` und `MailetContext`.

Das Interface `Mail` versieht eine MIME-Nachricht mit SMTP-Versandinformationen. Dazu gehören der Absender, (eventuell mehrere) Empfänger sowie weitere zusätzliche Informationen, beispielsweise die IP-Adresse des Versandrechners. `MailAddress` ist eine strengere Implementation von `InternetAddress` des JAVA-Standards JavaMail. Es bietet unter anderem Hilfsmethoden, um den Nutzeranteil und die Domain einer Emailadresse zu bestimmen. `MailetContext` bietet u.a. Zugriff auf die Verwaltung der Mailets.

## 2.4 MySQL

MySQL ist eine sehr schnelle, mehrbenutzerfähige und robuste Datenbank, welche der *General Public License* (GPL, Softwarelizenz) unterliegt. Damit ist es kostenlos für die private Nutzung und für Applikationen, bei denen der Quellcode mitgeliefert wird. Sowohl der vollständige Datenbank-Quellcode als auch Binärversionen für fast jedes Betriebssystem sind vorhanden. Genannt seien hier Linux, Windows 95/98/NT/2000/XP, Mac OS X und Solaris. Der Zugriff auf die eigentliche Datenbank – von Apache James aus – erfolgt mit Hilfe von *Structured Query Language* (SQL, strukturierte Abfragesprache) und *Java DataBase Connectivity* (JDBC, plattformunabhängige Schnittstelle zur Datenbanksoftware). Eine umfangreiche Dokumentation von MySQL ist in [14] zu finden. MySQL eignet sich damit hervorragend für die Datenhaltung.

## 2.5 HMAC-MD5

*Keyed-Hash Message Authentication Code* (HMAC, auf einer Hashfunktion basierender Integritätstest), definiert in RFC2104 [12], verwendet einen Schlüssel und die zu authentifizierenden Daten als Eingabeparameter für eine Hashfunktion. Die Sicherheit basiert dabei auf der verwendeten Hashfunktion. Bei der Realisierung *Message Digest 5* (MD5, kryptographische Prüfsumme), beschrieben in RFC1321 [13], erzeugt das Verfahren eine Prüfsumme von 128 bit. Sofern der private Schlüssel auf einem sicheren Weg ausgetauscht wurde, kann davon ausgegangen werden, dass die Daten authentisch sind. Bisher konnten noch keine erfolgreichen Angriffe gegen MD5 durchgeführt werden. HMAC-MD5 kann somit die Authentizität der *capabilities* sicherstellen.

## 2.6 Bestehende Lösungen gegen UCE

### 2.6.1 Filter auf Basis regulärer Ausdrücke

Diese Filter erkennen UCE anhand bestimmter Merkmale und sortieren ermittelten Spam aus. Dazu untersuchen sie Umschlag, Kopf und Rumpf jeder Mail. Mit Hilfe von Filterregeln, meist *regular expressions* (reguläre Ausdrücke), ist es vorher notwendig, die Charakteristik von UCE zu definieren. Beispielsweise filtert

```
/(spammer\.net|spammer\.com|spammer\.org)\?subject=remove/ REJECT
```

alle Emails, die als Betreff REMOVE beinhalten und von den Domains SPAMMER.NET, SPAMMER.COM oder SPAMMER.ORG abgesendet wurden. Die Filterregeln müssen ständig ergänzt oder verändert werden, um neue UCE ausfiltern zu können und haben damit nur eine bestimmte Trefferquote. Wenn eine Regel allerdings zu allgemein formuliert wird, besteht die Gefahr auch eine legitime Email als Spam zu klassifizieren.

### 2.6.2 Sperrung von Rechnern/Domainen

Diese Methode verwendet eine sogenannte *Realtime Blackhole List* (RBL, in "Echtzeit" aktualisierte Liste gesperrter Rechner). Es handelt sich dabei um ein zentral im Internet geführtes Verzeichnis, welches alle Absender enthält, die in der Vergangenheit als UCE-Versender tätig waren. Dabei wird die Liste periodisch aktualisiert, um auch neuere Spam-Absender erfassen zu können. Klienten, welche sich vor UCE-Versendern schützen möchten, müssen bei dem Empfang jeder Email eine Anfrage bei einem Listen-Server durchführen, um zu prüfen, ob der Absender gelistet ist. Dazu wird der *DNS reverse lookup* (Anfrage eines Domainnamens zu einer bekannten IP) verwendet. Ist die DNS-Prüfung beim Listen-Server positiv, wird die Annahme dieser Email verweigert. Nachteilig führt das zu einer Erhöhung der Netzlast, da für jede Verbindung eine zusätzliche DNS-Anfrage gestartet werden muss. Ein weiterer Nachteil ist, dass ein irrtümlich gelisteter Absender permanent von der Email-Kommunikation ausgeschlossen wird. Dieser kann keinen Einfluss auf den Inhalt des List-Servers nehmen. Problematisch sind daher vor allem dynamische Internetaddressen, wie sie zum Beispiel temporär an Nutzer von Einwahlknoten für die Dauer der Verbindung vergeben werden.

### 2.6.3 Bayes-Filter

[15] behandelt einen neuartigen statistischen Filteransatz. Dabei wird die gesamte Email, d.h. Kopf, Körper und Rumpf inklusive MIME-Anhang nach Wörtern durchsucht und ihre Anzahl in Tabellen gespeichert. Die erste Tabelle speichert Wort und Anzahl für legitime Nachrichten, die zweite für UCE. Eine weitere Tabelle enthält die berechnete Zuordnung zwischen den Wörtern und der Wahrscheinlichkeit ihres Auftretens in einer Spam-Nachricht. Neue Wörter erhalten eine geringe UCE-Wahrscheinlichkeit. Beim Eintreffen einer Email wird mit Hilfe eines angepassten Bayes-Algorithmus die Wahrscheinlichkeit ihrer Zuordnung als Spam berechnet.

Filter dieser Art erfordern ein bereits vorhandenes umfangreiches Repository legitimer bzw. Spam-Email des betreffenden Nutzers. Alternativ kann der Nutzer in einer Lernphase seine Email selbst in die entsprechenden Kategorien einordnen. Obwohl Bayes-Filter nach einer umfangreichen Einlernphase besser arbeiten als normale Filter, reagieren sie auf exotische Emails ebenfalls mit falschen Entscheidungen. Als Beispiel sei hier das beabsichtigte Verschicken einer Spam-Mail zu Anschauungszwecken (z.B. im Anhang) genannt. Im Gegenzug könnten Spammer ihre UCE-Email durch Verzicht auf typische UCE-Merkmale legitim formulieren.

### 2.6.4 Wegwerfadressdienste

Adressen für eine mehrmalige Verwendung können im Internet beispielsweise bei SpamGourmet [16] generiert werden. Bei der Registrierung legt der Nutzer die Anzahl der Nachrichten fest, die an die neue Adresse geschickt werden können. Optional kann eine Weiterleitungsadresse angegeben werden. Nach Ablauf des Nachrichtenzählers werden alle eingehenden Emails ignoriert. Wegwerfadressdienste sind insofern interessant, da man Identitäten beliebig erzeugen und auch wieder verwerfen kann. Ein typisches Anwendungsbeispiel ist das Ausfüllen eines Formulars im Internet, bei dem die Angabe einer Emailadresse obligatorisch ist.

### 2.6.5 Berichterstattungsdienste, Adressverfolgungsdienste

Unverlangte Werbung per Email ist nach überwiegender Rechtsauffassung in Deutschland illegal, wenn keine regelmäßige Geschäftsbeziehung besteht oder kein vorheriges ausdrückliches Einverständnis vorliegt. Seit Dezember 2001 ist diese Form der Werbung von der EU als rechtswidrig eingestuft worden. Die Empfänger von Spam können den Absender mit Hilfe von Adressverfolgungsdiensten zur Unterlassung auffordern oder ihn durch einen Anwalt abmahnen lassen. In der Realität können diese Dienste leider meist nicht genutzt werden, da der Versender aufgrund von gefälschten Adressen nicht ausfindig gemacht werden kann. Weitere problematische Faktoren sind Zeit- und Kostenaufwand und eine notwendige Kooperation zwischen den verschiedenen Netzbetreibern.

### 2.6.6 Teergruben

In [17] ist ein offensiver Ansatz, genannt “Teergrube”, beschrieben. Email-Versand erfolgt heute, wie auf Seite 5 erwähnt, hauptsächlich über SMTP. Dabei wird eine TCP/IP Verbindung zum MX-Host des betreffenden Empfängers hergestellt. Üblicherweise kann ein Rechner maximal ca. 65000 TCP/IP-Verbindungen gleichzeitig offenhalten. In der Regel sind es sogar weniger. SMTP bietet nun sogenannte Fortsetzungszeilen, mit denen die SMTP Sitzung offengehalten werden kann, ohne das ein *timeout* (Schwellwert für aktivitätslose Zeitspanne) erfolgt. Ein SMTP-Rechner sendet als Antwort auf die Kommandos des Klienten Zeilen, die aus einem Fehlercode, einem Leerzeichen und einem für Menschen

lesbaren Text bestehen. Wird das Leerzeichen durch ein Minuszeichen ersetzt, ist der Rechner noch nicht mit der Antwort fertig. Damit gelingt es, einen Port bei der Mailauslieferung offen zu halten – idealerweise über mehrere Stunden hinweg. Somit reduziert sich in dieser Zeit die Leistungsfähigkeit des UCE-Senders. Derartige Fortsetzungszeilen werden im Abstand von Minuten gesendet. Dies verbraucht minimale Bandbreite und stoppt trotzdem den Versender wirksam. Nachteil des Verfahrens ist, dass ein UCE-Versender nach Erhalt einer Fortsetzungszeile den Versand sofort abbrechen kann. Er könnte auch eine grosse Anzahl von Emails parallel an diesen Server verschicken.

### 2.6.7 Absenderverifikation mit SMTP

Das Protokoll SMTP sieht keine Zugriffsbeschränkung auf den SMTP-Server vor. Das Protokoll POP hat dagegen eine Authentifikationsmöglichkeit in Form eines Nutzernamens und dem zugehörigen Passwort. Um sicherzustellen, dass nur angemeldete Nutzer Emails versenden können, muss der Nutzer sich zunächst per POP anmelden und kann dann für eine festgelegte Zeit SMTP benutzen. Diese Methode wird kurz als *SMTP-after-POP* bezeichnet. Probleme bereitet diese Methode den Nutzern von MUA's, die keine Konfigurationsmöglichkeiten von SMTP-after-POP anbieten. Genannt sei hier die Benutzung von Microsoft Outlook, das eine Erstellung von 2 Ordnern – jeweils für Versand und Empfang – und eine serielle Abfrage dieser von Hand erfordert.

### 2.6.8 Tagged Message Delivery Agent (TMDA)

TMDA[18] ist – bis auf einige abweichende Details – eine Implementation vom ersten Schritt des in diesem Beleg vorgestellten Verfahrens. Es kombiniert eine "weisse Liste" – also eine Liste aller bekannten und vertrauenswürdigen Absender – mit einer "schwarzen Liste" für unerwünschte Absender. Des weiteren ist ein automatischer Antwortmechanismus implementiert, der von einem unbekannten Absender eine Bestätigungs-Email erhalten muss, bevor weitere Nachrichten akzeptiert werden. TMDA agiert gleichzeitig als MDA mit einer flexiblen Filtersprache, die eine genaue Kontrolle über eingehende und ausgehende Mail ermöglicht. Die Strategie lautet: "Alles verbieten, was nicht explizit erlaubt ist.".

Eine Beispielkonfiguration von TMDA für eingehende Emails könnte so aussehen:

```
### ~/.tmda/filters/incoming (first match wins) ###
# Accept all bounces (messages with an empty envelope sender)
from <> ok
# Accept all messages to postmistress
to postmistress@* accept
# Bounce all messages from badboy.dom
from *@=badboy.dom bounce
# Include my blacklist and whitelist
```

```
from-dbm ~/.tmda/lists/blacklist.db drop
from-cdb ~/.tmda/lists/whitelist.cdb accept
from-file -autodbm ~/.tmda/lists/nastygrams bounce
from-file -autocdb ~/.tmda/lists/confirmed ok
from-file ~/.tmda/lists/whitelist_wildcards accept
# Revoked addresses
to jason-stupid_promo.289076@mastaler.com bounce
to jason-jcrew.832234@mastaler.com confirm
# Examine the message content
body "viagra|ginseng" confirm
headers 'Precedence:.*junk' reject
headers -case 'MAKE MONEY FAST' drop
# Accept all messages smaller than 10KB,
# but drop messages larger than 1MB
size <10000 deliver
```

Eine eingehende Email wird auf die formulierten Bedingungen geprüft. Der erste passende Eintrag entscheidet, wie mit der Email weiter zu verfahren ist. Obiges Beispiel zeigt nur einen Ausschnitt aus einer üblichen Konfiguration, dies zeigt das Ausmaß der erforderlichen Komplexität einer Arbeitskonfiguration.

Da TMDA als MDA arbeitet, ist es direkt abhängig vom MTA. Bisher wurde Unterstützung für sendmail, qmail und postfix[19] realisiert. Damit gestaltet sich die Portabilität auf andere Systeme kompliziert beziehungsweise wird ohne Programmierarbeit unmöglich. TMDA ist in Phyton implementiert und daher beschränkt plattformunabhängig. Die Installation des Paketes sowie eine nachträgliche Administration sind schwierig und zeitaufwändig.

### 2.6.9 Vergleich aller Verfahren

In nachstehender Tabelle werden die vorgestellten Verfahren in Bezug auf Wirksamkeit gegen UCE, Komplexität in der Konfiguration und Administration, Benutzbarkeit für einen Nicht-Experten, Portabilität und Größe des Einsatzfeldes verglichen:

| Verfahren | UCE-Wirks. | Konfig. | Benutzb. | Port. | Einsatzfeld |
|---|---|---|---|---|---|
| Filter (reg. Ausdrücke) | mittel | hoch | niedrig | hoch | hoch |
| Sperrung von Rechnern | mittel | mittel | – | hoch | hoch |
| Bayes-Filter | hoch | hoch | niedrig | mittel | niedrig |
| Wegwerfadressdienste | hoch | niedrig | hoch | hoch | niedrig |
| Adressverfolgungsdienste | niedrig | mittel | hoch | mittel | niedrig |
| Teergruben | hoch | niedrig | – | hoch | niedrig |
| Absenderverifikation | mittel | mittel | mittel | hoch | mittel |
| TMDA | hoch | hoch | mittel | mittel | mittel |

## 2.7 Zyklenopfer

Die Voraussetzung für die Umsetzung des zweiten Entwurfsschrittes ist eine Methode, dem Versender von Emails einen abschätzbaren Aufwand zu erzeugen. Eine geeignete Vorgehensweise ist die Erzeugung von Rechenaufwand – also Zyklenopfer. Verschiedene Arten der Zyklenopfer sind in [20] beschrieben, darunter auch ein hashbasiertes Verfahren, welches in einer neueren Entwicklung HashCash [21] verwendet wurde. Die Grundidee ist bei allen Verfahren gleich: Es sollen CPU-Zyklen verbraucht und damit ein Aufwand für den Versand von Email erzeugt werden. Versender von Massenmail benötigen somit eine extrem hohe, d.h. unerreichbare Rechenleistung. Urbilder von Hashfunktionen sind nur mit exponentiellem Aufwand wieder rekonstruierbar. Damit sind sie für eine genügend große Länge aufwändig zu berechnen, können aber mit kleinem Aufwand geprüft werden. Die benötigte Rechenzeit ist dabei von der Bitlänge abhängig. Jedes weitere Bit erhöht den Rechenaufwand im Durchschnitt um das Doppelte. Mobile Geräte mit geringer Rechenleistung, wie PDAs oder Mobiltelefone können durch die Vergabe von “Gutscheinen” durch den Empfänger unterstützt werden.

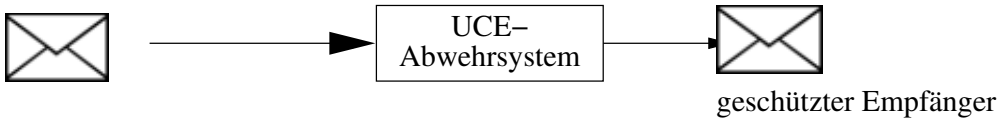
# 3 Entwurf

## 3.1 Verfahren

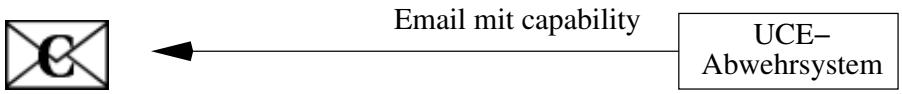
### 3.1.1 Erster Schritt

Um einen wirkungsvollen Schutz gegen UCE zu erreichen, wird die gesamte Kommunikation zwischen Absender und Empfänger von einer Zwischenschicht auf Seite des Empfängers abgefangen. Diese Zwischenschicht ist nun in der Lage, automatisch eine Rückantwort zu generieren. Die Antwort enthält eine *capability* (Fähigkeit), welche die zukünftige Email-Kommunikation mit dem Empfänger ermöglicht.

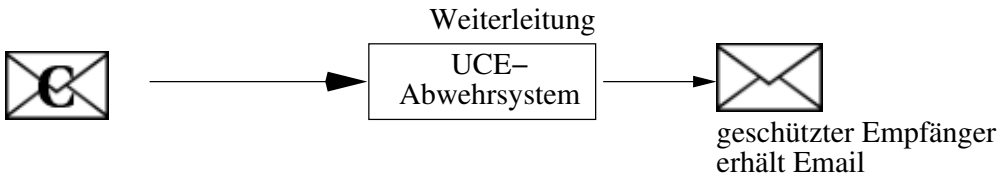
1. unbekannter Versender schickt Email an geschützten Empfänger



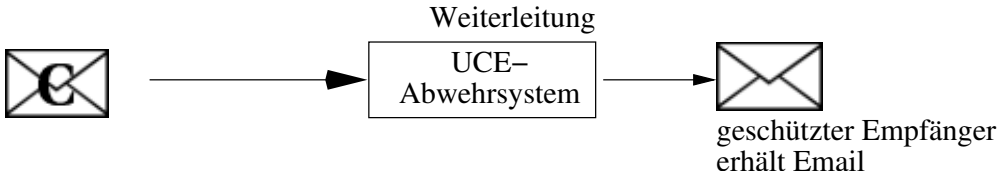

2. unbekannter Versender erhält capability




3. unbekannter Versender verschickt Email erneut mit capability




alle weiteren Emails: unbekannter Versender verschickt Email mit capability




Dabei hat der automatische Antwortmechanismus folgende Aufgaben:

- Erzeugung einer *capability*, welche in das FROM und das REPLY TO-Feld der Email eingefügt wird
- Senden einer automatischen Rückantwort auf jede Email, außer auf solche, die eine gültige *capability* besitzen

- Speichern einer Liste aller gültigen capabilities mit der entsprechenden Zuordnung zu Personen bzw. Institutionen in einer Datenbank

Heutzutage akzeptieren die meisten UCE-Versender keine Antworten oder die Absenderadresse ist gefälscht. Daher müssen sämtliche Emails, die externe Mailsystemen aufgrund eines nicht existenten Nutzeranteils zurückweisen, verworfen werden.

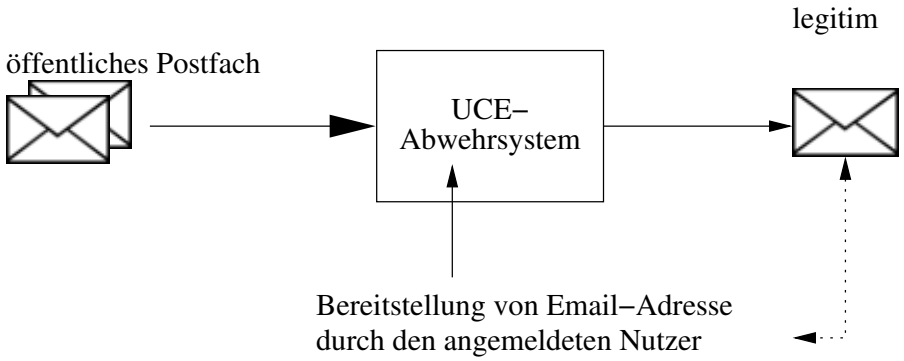
Dieser erste Schritt gegen UCE ermöglicht es, sämtliche Emails, deren Absender keine Antworten akzeptieren, abzuwehren. Wenn dieser Schritt so nutzerfreundlich in vorhandene Mailsysteme integriert werden kann, dass er eine gewisse Verbreitung findet, wird ein UCE-Versender, der vorher mit Leichtigkeit Millionen von Emails versendet hat, Millionen von Antworten erhalten. Damit wird UCE teuer für den Versender.

Bei diesem Ansatz sind Sonderfälle zu beachten. Der UCE-Versender kann mit geringer Wahrscheinlichkeit die *capability* und den passenden zugehörigen Absender herausfinden. Hier muss die *capability* entzogen werden. Die Quelle der UCE wird anhand der Zuordnungen zwischen Absender und *capability* offenbar. Der entsprechende Eintrag wird aus der Datenbank entfernt.

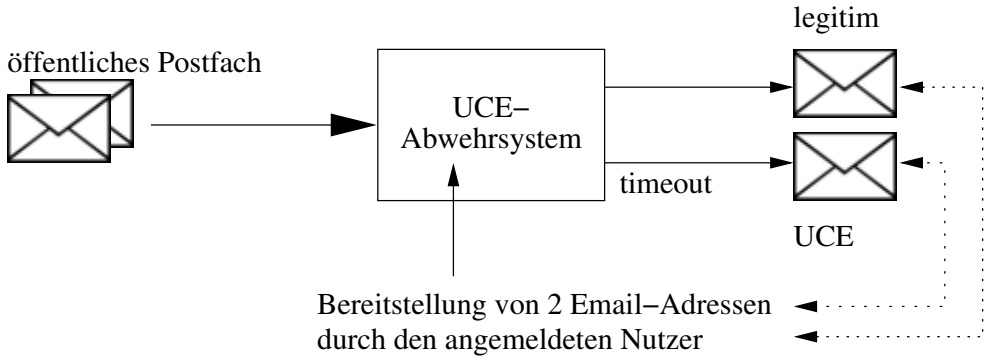
Problematisch sind Mailinglisten wegen der Diskrepanz zwischen Absender- und Antwortadresse. Durch automatisch generierte Antworten, die auf der Mailingliste erscheinen, würde man selbst zum Spammer. Denkbar sind hier zwei alternative, aber komplett gegensätzliche Ansätze. Zum einen kann man sich eine *capability* generieren, diese in die Datenbank eintragen und die Registrierung der Mailingliste mit *capability* durchführen. Zum zweiten könnte man die Mailingliste für den korrekten Umgang mit capabilities modifizieren.

### 3.1.2 Verbesserung der Skalierbarkeit

Um eine gute Skalierung bei einem Betrieb auf einem zentralen Mailserver auch bei großen Nutzerzahlen zu gewährleisten, ist das Konzept als Weiterleitungssystem konstruiert. Emails werden nicht lokal gespeichert und abgefragt, sondern bei der Registrierung wird eine Emailadresse angegeben, zu der legitime Nachrichten weitergeleitet werden.




Eine zukünftige Erweiterung auf die Registrierung von 2 Emailadressen könnte wie folgt realisiert werden:




An die erste Adresse werden legitime Nachrichten inklusive *capability* gesendet. An letztere werden Emails weitergeleitet, die nach einer festgelegten Zeitspanne keine Bestätigung erhielten und damit mit hoher Wahrscheinlichkeit UCE sind.

### 3.1.3 Geplanter zweiter Schritt

Das im ersten Schritt vorgestellte Verfahren wird nur so lange funktionieren, bis der UCE-Versender antwortfähig wird. Dazu muss in einem zweiten Schritt der Erhalt einer *capability* erschwert werden. Dies könnte zum Beispiel in Form von Zyklenopfern erfolgen. Die erste automatische Rückantwort enthält dann nicht die *capability* im Klartext, sondern in Form einer aufwändigen mathematischen Rechenoperation. Beispiele sind die auf Seite 13 genannten Zyklenopfer-Verfahren.

## 3.2 Ansatzpunkt der Filterung

Der automatische Antwortmechanismus muss in der Lage sein, sämtliche Kommunikation zum Empfänger abzufangen. Dies kann *clientseitig* (auf Seite des Empfängers) oder *serverseitig* (auf einem zentralen Schutzserver) geschehen. Empfängerseitig sind folgende Eigenschaften zu beachten:

- Es handelt sich um eine eigenständige Anwendung, welche dauerhaft im Hintergrund ausgeführt wird.
- Der Klient sollte eine ständige Netzverbindung haben oder die Emailadressen müssen alle auf eine zentrale Adresse kanalisiert werden. Dies ist zwar technisch möglich, aber aufwändig oder oft gar nicht zu erreichen. (Beispielsweise, wenn die Emailadresse von einem privaten Email-Anbieter vergeben wurde.)
- Das System selbst ist betriebssystemunabhängig. Bei mehreren Betriebssystemen auf ein- und demselben Rechner muss es in jedem Betriebssystem neu installiert werden. Die Datenbankinformationen benötigen in diesem Fall eine gemeinsam genutzte Ressource, um Konsistenz zu gewährleisten.
- Internet-Mail, wie z.B. die Abfrage von Emails über ein Webinterface von einem fremden Rechner aus, kann nicht geschützt werden, da der UCE-Schutz nur auf dem entsprechenden Klienten ermöglicht werden kann.

Der Betrieb auf einem zentralen Serverrechner bietet folgende Eigenschaften:

- Dienst auf einem dedizierten Serverrechner.
- Es besteht keine Notwendigkeit der Installation von zusätzlicher Software auf dem Endrechner, dafür ist aber eine Registrierung für den Dienst am Server erforderlich.
- Probleme der Skalierbarkeit, wenn Emails aller Nutzer zentral auf dem Server aufbewahrt werden sollen.

## 3.3 Capability-Sicherung

Die Sicherung der *capability* in Bezug auf Fälschung oder Veränderung erfordert ein kryptographisch starkes Verfahren. Das auf Seite 8 beschriebene HMAC-MD5-Verfahren bietet die benötigten Eigenschaften.

## 3.4 Datenhaltung

Die Entwurfsentscheidung über die Datenhaltung wird im wesentlichen durch Flexibilitäts- und Sicherheitsbetrachtungen beeinflusst. Dies legt die Nutzung eines Datenbanksystems nahe. Die Vorteile einer Datenbankanwendung liegen nicht nur in der möglichen Komplexitätsbeherrschung sondern auch in der integrierten Transaktionsverwaltung, d.h. der möglichen Abwicklung gleichzeitiger Zugriffe mehrerer Benutzer. Weitere Vorteile sind Zugriffsrechte, Datenkompatibilität sowie Fernzugriff.

# 4 Implementierung

## 4.1 Vorbemerkung

In diesem Kapitel soll ein lauffähiges System vorgestellt werden, im folgenden *SpamStop* genannt. Es beinhaltet den ersten Schritt des Entwurfes.

## 4.2 Filterung

SpamStop ist als Aufsatz auf einen kompletten Mailserver implementiert. Der entscheidende Vorteil der Nutzung eines Mailservers im Vergleich zu anderen Lösungen ist die Einführung einer generischen Abstraktionsschicht zwischen MTA und MUA. Der auf Seite 6 beschriebene Mailserver James fand dafür Verwendung.

Zum Abfangen der gesamten Kommunikation wurde zunächst ein minimaler Matcher `SpamStopMatcher.java` implementiert und in die Verarbeitungskette eingefügt. Alle Emails werden somit an das SpamStop-Mailet `SpamStopMailet.java` weitergereicht:

```
public boolean matchRecipient(MailAddress recipient) {
        /* intercept everything */
        return true;
}
```

Der entsprechende Konfigurationseintrag in der Datei `config.xml` lautet:

```
<mailet match="SpamStopMatcher" class="SpamStopMailet">
```

Dadurch ist das Ziel des Entwurfes, die gesamte Kommunikation abzufangen, erfüllt.

## 4.3 Automatischer Antwortmechanismus

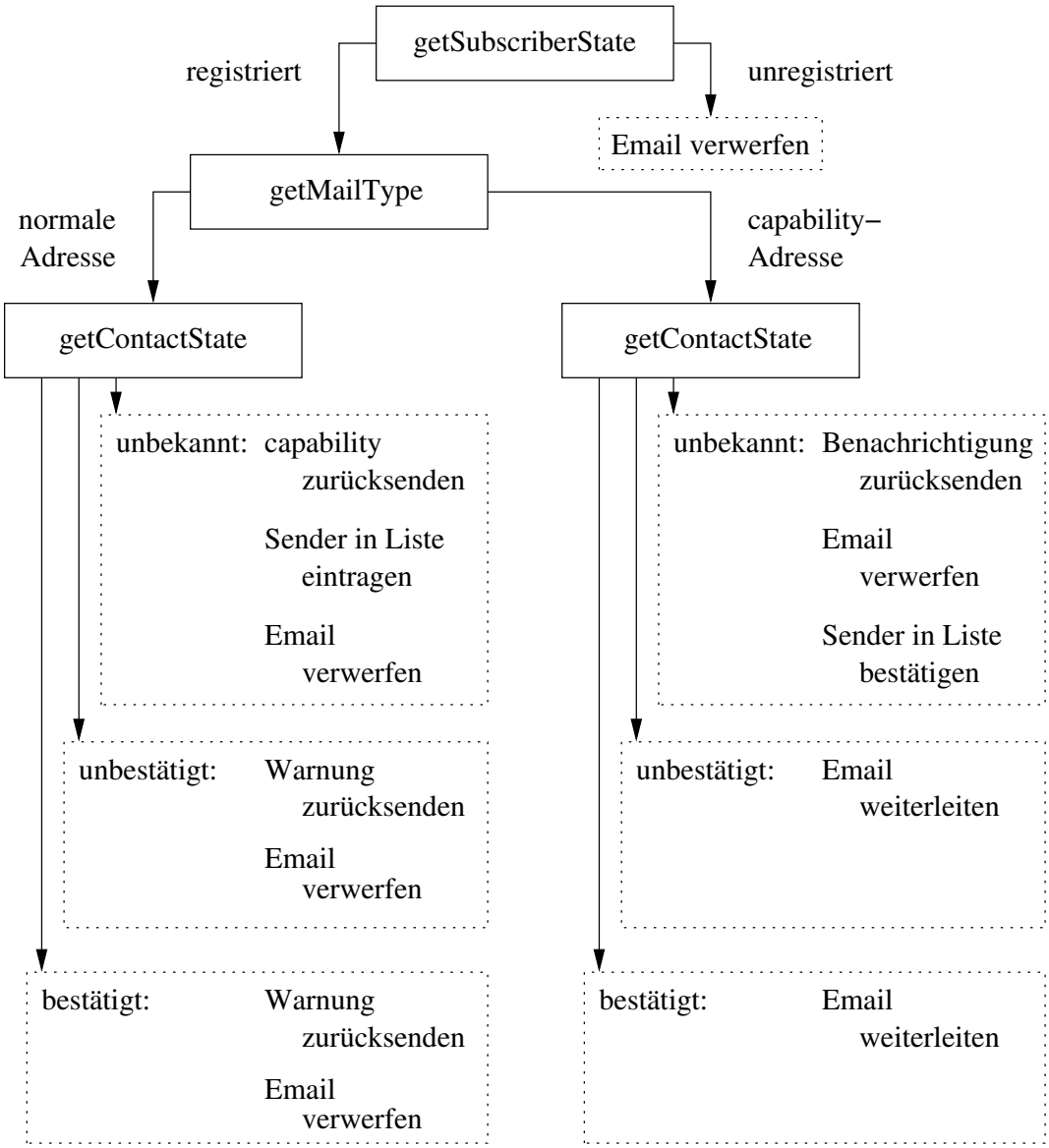
Die auf Seite 14 dargelegten Aufgaben eines automatischen Antwortmechanismus erfüllt das SpamStop-Mailet. Folgende Routinen übernehmen die Identifikation eingehender Emails:

```
getSubscriberState(String recipientEmailAddress)
getMailType(String recipientEmailAddress)
getContactState(String recipientEmailAddress,
                String senderEmailAddress)
```

Die Methode `getSubscriberState(String recipientEmailAddress)` ermittelt, ob der Empfänger sich für SpamStop beim Server angemeldet hat. Mögliche Rückgabewerte sind REGISTRIERT oder UNREGISTRIERT. `getMailType` ermittelt den Typ der Mail, d.h. ob es sich um eine Mail mit gültiger *capability* handelt oder nicht. Rückgabewerte sind hier NORMALE ADRESSE oder CAPABILITY-ADRESSE. `getContactState(String`

`recipientEmailAddress, String senderEmailAddress)` überprüft den Status des Absenders innerhalb der Adressdatenbank in Bezug auf die Absenderadresse. Gültige Rückgabewerte sind UNBEKANNT, UNBESTÄTIGT und BESTÄTIGT.

Mit Hilfe dieser 3 Methoden lassen sich alle Möglichkeiten der Kommunikation modellieren:



## 4.4 Capability-Sicherung

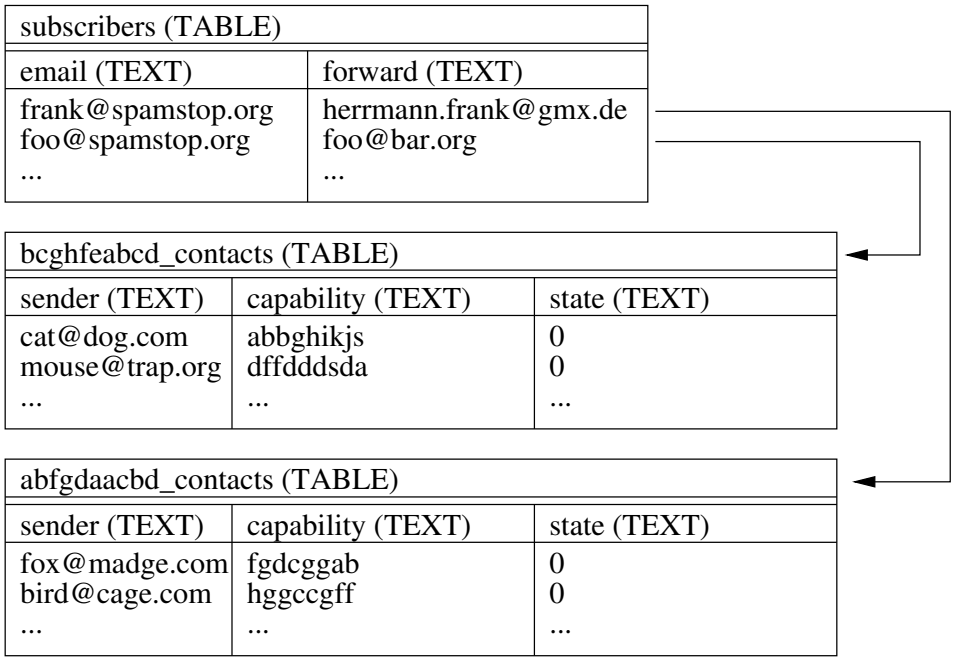
JAVA enthält bereits die erforderlichen Klassen für die Benutzung des HMAC-MD5-Verfahrens:

```
/* necessary classes */
import java.security.*;
import javax.crypto.*;

public class testMAC{

    public static void main(String[] argv) throws Exception {

       /* create secret key for hmac md5 */
       KeyGenerator kgen = KeyGenerator.getInstance("HmacMD5");
       SecretKey key = kgen.generateKey();

       /* get mac instance and initialize with secret key */
       Mac hmac= Mac.getInstance("HmacMD5");
       mac.init(key);
       byte[] result = mac.doFinal("Nachricht".getBytes()));
    }
}
```

Dieses Beispiel demonstriert die einfache Verwendung der Pakete `java.security` und `javax.crypto`. Ein geheimer Schlüssel wird erzeugt und damit ein HMAC-MD5-Objekt initialisiert. Mit diesem Schlüssel und den zu verschlüsselnden Daten (NACHRICHT) wird der resultierende Hashwert der Variable RESULT zugeordnet.

## 4.5 Datenhaltung

Alle registrierten Nutzer werden in die Tabelle SUBSCRIBERS aufgenommen, bestehend aus der registrierten (neuen) Emailadresse und der vom Nutzer gewünschten (alten) Weiterleitungsadresse:




Für jeden registrierten Nutzer legt SpamStop eine separate Tabelle an, in der die Kontakte gespeichert werden. Der Tabellenname setzt sich aus einem eindeutigen Hashcode, einem Unterstrich und dem String CONTACTS zusammen. Dieser Aufwand ist notwendig, da im Allgemeinen in Tabellennamen keine Sonderzeichen verwendet werden dürfen bzw. diese spezielle Bedeutungen haben. Das verwendete Schema gewährleistet Portabilität auf andere Datenbanken.

Die Datenquelle wird für den Zugriff auf MySQL vorbereitet, indem im James-Konfigurationsblock `config.xml` folgende Einträge vorgenommen werden:

```
<!-- The database-connections block -->
<database-connections>
   <data-sources>
       <data-source name="james"
          class="org.apache.james.util.mordred.JdbcDataSource">
           <driver>org.gjt.mm.mysql.Driver</driver>
           <dburl>jdbc:mysql://127.0.0.1/james</dburl>
           <user>root</user>
           <password>test</password>
           <max>2</max>
       </data-source>
   </data-sources>
 </database-connections>
```

Als JDBC-Treiber wird ORG.GJT.MM.MYSQL.DRIVER benutzt, die Adresse lautet JDBC:-MYSQL://127.0.0.1/JAMES, d.h. die Datenbank läuft auf 127.0.0.1, dem lokalen Rechner

und ihr Name ist JAMES. Im vorliegenden Konfigurationsfall wird MySQL als Administrator mit dem Passwort TEST angesprochen und die maximale Anzahl paralleler Verbindungen ist 2.

## 4.6 Konfiguration

Die folgende Tabelle beschreibt eine Auflistung aller relevanten Konfigurationsparameter aus `config.xml`:

| Wert | Bedeutung |
|---|---|
| postmaster | Administratoradresse |
| administrator_accounts | Administratorzugang |
| dnsserver | Adresse des DNS-Servers |
| headerText | Kopftext, der jeder Mail von SpamStop vorangestellt wird |
| footerText | Fußtext, der jeder Mail von SpamStop angehangen wird |
| requestText | Text, der unbekannten Absendern zugesandt wird |
| warningText | Text, der eingetragenen Absendern zugesandt wird, die wiederholt ohne entsprechende *capability* Email versenden |
| notificationText | Benachrichtigungstext, der zugesandt wird, wenn eine *capability* nicht mehr existent ist |
| hashAlgorithm | verwendeter Hash-Algorithmus |
| secretKeyBytes | privater Schlüssel für den Hash-Algorithmus und Administration |
| data-sources | verwendete Datenbank-Ressource |

## 4.7 Administration

Während der Arbeit an SpamStop entstand die Idee, Administration direkt über das Abschicken einer Email an die Administratoradresse des betriebenen Servers zu ermöglichen. Dies erlaubt ein hohes Maß an Flexibilität. Die Authentifikation erfolgt über ein Passwort, welches nur dem Administrator bekannt ist und dem Betreff der Email – abgetrennt durch einen Stern (*) – enthalten ist. Die (vorläufigen) Befehle lauten:

| Befehl | Funktion |
|---|---|
| list | listet alle registrierten Nutzer |
| add <email1>#<email2> | neuer Nutzer <email1> und Weiterleitungsadresse <email2> |
| del <email> | löscht Nutzer <email> |
| reset | löscht alle Nutzer und Kontaktlisten |
| help | sendet eine Hilfenachricht |

## 4.8 Beispielszenario

### 4.8.1 Ausgangssituation

Der SpamStop-Server ist auf den Domainnamen `spamstop.org` registriert. Die Administratoradresse ist als `postmaster@spamstop.org` konfiguriert.

### 4.8.2 Registrierung

Der Administrator mit der Email-Adresse `admin@irgendwo.org` registriert die neue Adresse `nutzer@spamstop.org`, der seine Mails an `bsp@inf.tu-dresden.de` weiterleiten lassen moechte. Das Passwort lautet im Beispiel GEHEIM. Daher sendet er folgende Mail:

```
From: admin@irgendwo.org
To: postmaster@spamstop.org
Subject: add nutzer@spamstop.org#bsp@inf.tu-dresden.de*geheim
Body: (leer)
```

SpamStop schickt eine Bestätigung an den Administrator:

```
From: postmaster@spamstop.org
To: admin@irgendwo.org
Subject: Re: add nutzer@spamstop.org#bsp@inf.tu-dresden.de*geheim
Body:
nutzer@spamstop.org with forward bsp@inf.tu-dresden.de has
been added with hash pfofhobbbd
```

### 4.8.3 Kommunikation

Der Email-Nutzer `fremd@fremd.de` möchte Kontakt mit `nutzer@spamstop.org` aufnehmen:

```
From: fremd@fremd.de
To: nutzer@spamstop.org
Subject: Hi!
Body: Missed you so much.
```

SpamStop erkennt, dass der Absender bisher unbekannt ist, vergibt eine *capability* und schickt eine Mail zurück:

```
From: nutzer-hashed-hoepgh@spamstop.org
To: fremd@fremd.de
Subject: Re: Hi!
Body:
Because you have not been in contact with
the recipient before, you have to
```

```
send this message again by pressing the
reply button of your mail software NOW.
Store my new email address in your address book!

Attachment: (original message)
```

Nach einer erfolgreichen erneuten Sendung werden alle Emails an die *capability*-Adresse *nutzer-hashed-hoepgh@spamstop.org* an `bsp@inf.tu-dresden.de` geschickt.

### 4.8.4 Administration

Ein Beispiel sei das Anzeigen aller bisher registrierten Nutzer. Dazu sendet der Administrator folgende Email:

```
From: admin@irgendwo.org
To: postmaster@spamstop.org
Subject: list*geheim
Body: (leer)
```

Als Antwort sendet SpamStop:

```
From: postmaster@spamstop.org
To: admin@irgendwo.org
Subject: Re: list*geheim
Body:
subscribers:

nutzer@spamstop.org->bsp@inf.tu-dresden.de
```

# 5 Bewertung

## 5.1 Einsatz

SpamStop ist im Moment noch sehr spartanisch zu bedienen und hat enormes Potential für Verbesserungen hinsichtlich Nutzbarkeit und Komfort. Trotzdem ist es bereits eine robuste Lösung gegen UCE. Der darunterliegende Mailserver James wird ständig weiterentwickelt und befindet sich mittlerweile in Version 2.1 auf der James-Entwicklerseite. Insbesondere an IMAP-Unterstützung und einem Tool zur Leistungsmessung wird derzeit gearbeitet. Seitens der James-Entwickler ist ausserdem eine Internetseite geplant, die speziell Mailets gewidmet ist. Sie wird voraussichtlich unter `www.mailet.org` in den nächsten Wochen erscheinen.

## 5.2 Praktische Tests

Für erste Tests eines praktischen Einsatzes der Software wurde ein Rechner mit Pentium Pro-Prozessor (200 MHz) und 64 MB RAM verwendet. Das Java Software Development Kit in Version 1.4, sowie Apache James Version 1.2.2 bildeten die Entwicklungsgrundlage. MySQL Version Version 9.38, Distribution 3.22.32 kam als Datenbank zum Einsatz. Sowohl James als auch MySQL arbeiteten als dedizierte Serverdienste. Sie zugrundeliegende Programmiersprache JAVA ist um den Faktor 10 bis 20 langsamer als eine vergleichbare Hochsprache wie C. Daher wurde zunächst die Leistung von SpamStop mit möglichst hohen Belastungen geprüft, um Praxistauglichkeit sicherzustellen. Für den Versand der Nachrichten zum SpamStop-Server wurde ein universitätsexterner Rechner mit AMD Athlon Prozessor (800 Mhz) und 512 MB RAM verwendet. Als Internetverbindung stand ein 10 MBit Zugang zur Verfügung. Linux, Kern 2.4.19 und das Kommandozeilenwerkzeug *mail* übernahmen den Versand der Emails. Folgendes *Bourne-Again SHell Skript* (Bash, Kommandozeileninterpreter) versendete die Emails:

```
export POSTMASTER="postmaster@albrecht.inf.tu-dresden.de"
export GAP=2
# reset the database
mail -s "reset" $POSTMASTER <"."
# wait to be sure reset transaction has been finished
sleep $GAP
# add user $1 times (according to
# specified parameter) with forward address
for i in `seq 1 $1`
do
mail -s "add $i@albrecht.inf.tu-dresden.de#test@test.test" \
$POSTMASTER <"."
done
```

Obenstehendes Skript wurde mit dem Bashbefehl *time* (Werkzeug zur Messung von Ausführungszeiten) aufgerufen, um die Sendezeit zu erhalten. Die Sendezeit ist die Zeit, die für den Versand Nachrichten tatsächlich benötigt wird. Die Antwortzeiten – d.h. die Zeit, die verstrichen ist, bis alle Antworten zugestellt sind – wurden anhand der Zeitstempel des SpamStopMailets berechnet. Es entstanden folgende Messergebnisse:

| Anzahl der Emails | Sendezeit (in s) | Antwortzeit (in s) |
|---|---|---|
| 5 | 0,1 | 4 |
| 10 | 0,2 | 8 |
| 20 | 0,4 | 15 |
| 50 | 0,9 | 34 |
| 100 | 1,8 | 75 |
| 200 | 3,5 | 282 |
| 500 | 7,5 | 589 |
| 550 | * | * |

Wie zu erwarten, ist der Versand von Email linear im Aufwand. Die Antwortzeit verhält sich zunächst linear, bei steigender Nachrichtenanzahl erhöht sich aber die Antwortzeit gravierend.Die Ursache sind hier wahrscheinlich die Leistungsdaten des verwendeten Testrechners. Insbesondere mehr Hauptspeicher und ein schneller Prozessor würden hier Abhilfe schaffen. Da die Nachrichten ohne Verzögerung hintereinander versendet wurden, kam es ab ca. 550 Nachrichten zu einem Mangel an Ressourcen beim Versandrechner – es konnten durch den MTA keine weiteren Prozesse geforkt werden.
Der Test über ein Webportal, *Global Mail Exchange* (GMX, Email-Provider) www.gmx.de, konnte nicht erfolgreich durchgeführt werden. Keine Emails, die SpamStop versendete, wurden angenommen. Die Emails wurden weder zurückgebounct, noch zugestellt. Die Vermutung einer für den Nutzer nicht beeinflussbaren Anti-Spam bzw. Anti- bounce-Maßnahme lag nahe. Eine entsprechende Anfrage an die GMX AG blieb unbeantwortet. Der SpamStop-Server ist, da durch eine Firewall geschützt, vom Internet aus nicht direkt erreichbar. Dies könnte höchstwahrscheinlich die Ursache für das Verhalten sein.

# 6 Zusammenfassung und Ausblicke

Die vorliegende Arbeit beschreibt den Entwurf und eine erste praktische Realisierung einer *capability*-basierten Lösung gegen UCE. SpamStop, das als Ergebnis dieses Belegs entstand, ist eine erste lauffähige Demonstration dieses Verfahrens. Für weiterführende Projekte bieten sich folgende Zielstellungen an:

- **Benutzerfreundlichkeit** – Das System kann nur einen hohen Verbeitungsgrad erreichen, wenn es für jeden Emailnutzer verwendbar ist. Daher sollte die Verbesserung der Benutzerschnittstelle höchste Priorität erhalten. Dazu gehört die Erstellung einer webbasierten Registrierungsmöglichkeit ohne Mithilfe eines Administrators, eine intuitive Antwortmail und eine motivierende Textnachricht für potentielle Neubenutzer, die eine Antwortmail erhalten.

- **Zyklenopfer** – Auf Seite 13 wurde die Idee eines erschwerten Erhalts von capabilities beschrieben. Diese könnte in SpamStop integriert werden, wobei insbesondere auf die praktische Umsetzung unter Beachtung der Vielfalt existierender MUA's eingegangen werden muss.

- **Adresstypen** – Bisher gibt es lediglich den Adresstyp *hashed*, der dem Benutzer das Vorhandensein einer *capability* signalisiert. In Betracht kommen aber auch weitere Adresstypen, wie sie zum Beispiel bei Wegwerfadressdiensten benutzt werden. Also beispielsweise *countdowned* für Adressen, deren Nachrichtenanzahl begrenzt ist, oder *expires* für Adressen, deren Gültigkeitsbereich nur innerhalb einer gewissen Zeitspanne festgelegt ist.

- **Optimierung** – Bei der Erstellung der Software wurde Wert auf Funktionalität und Fehlerfreiheit gelegt, jedoch blieb der Optimierungsaspekt bisher unberücksichtigt. Potential für Effizienzsteigerung ist vor allem bei den Datenbankabfragen und bei der Java Virtual Machine vorhanden. Eventuell kann der Einsatz eines *Just In Time compilers* (JIT compiler, plattformabhängiger Binärübersetzer) dazu in Betracht gezogen werden.

- **Protokollierung** – Bisher können Statusmeldungen nur auf der Konsole verfolgt werden. Eine umfangreiche Protokollierung auf Festspeicher wäre wünschenswert.

# A Glossar

| | |
|---|---|
| API | *Application Programming Interface*, Programmierschnittstelle |
| ASCII | *American Standard Code for Information Interchange*, 7 bit Zeichenkode |
| DNS | *Domain Name System*, verteilter Namensdienst |
| IMAP | *Interactive Mail Access Protocol*, interaktives Zugriffsprotokoll, im Gegensatz zu POP3, zustandsbehaftetes Protokoll zum Abholen der Emails von einem Mailserver |
| JAMES | *Java Apache Mail Enterprise Server*, Mailserver von Apache |
| JDBC | *Java DataBase Connectivity*, plattformunabhängige Schnittstelle zur Datenbanksoftware |
| JSSE | *JServ Servlet Engine*, JAVA Servletumgebung |
| MDA | *Mail Delivery Agent*, Zustellungssystem, nimmt lokal Emails an und verteilt sie in die Mailboxen der Nutzer |
| MIME | *Multipurpose Internet Mail Extensions*, universale Email-Erweiterung, die dem RFC822-Standard weitere Felder hinzufügt, ermöglicht u.a. Anhänge und Zeichensätze |
| MTA | *Mail Transfer Agent*, Mail Transportsystem, sorgt für die Auslieferung der Emails an die vorgesehenen Rechner |
| MTS | *Message Transfer System*, Nachrichtentransportsystem |
| MUA | *Mail User Agent*, Email-Benutzerprogramm, zeigt dem Nutzer die Emails an |
| MX record | *Mail Exchange record*, DNS-Eintrag für Email |
| NNTP | *Net News Transfer Protocol*, Transportprotokoll für Diskussionsforen |
| PGP | *Pretty Good Privacy*, Verschlüsselungssoftware für Emailkommunikation |
| POP3 | *Post Office Protocol Version 3*, Empfangsprotokoll Version 3, im Gegensatz zu IMAP, zustandsloses Protokoll zum Abholen der Emails vom Mailserver |
| Provider | Servicedienst, der gegen Gebühren die technischen Voraussetzungen für den Anschluss ins Internet ermöglicht |
| RBL | *Realtime Blackhole List*, periodisch aktualisierte Listen gesperrter Rechner |
| SMTP | *Simple Mail Transfer Protocol*, Protokoll für den Versand von Email |
| SPAM | Akronym für *SPiced HAM*, geschützter Markenname der Hormel Foods Corporation für Würzfleisch |
| UCE | *Unsolicited Commercial Email*, unerwünschte Werbemail |

# B Literaturverzeichnis

## Literatur


[01] *TU Berlin Hoax-Info Service*,
http://www.tu-berlin.de/www/software/hoax.shtml

[02] *RFC 822: Standard for the Format of ARPA Internet Text Messages*, 1982,
ftp://ftp.tu-dresden.de/pub/documents/rfc/rfc822.txt

[03] *Sendmail Home Page*, www.sendmail.org

[04] *qmail: Second most popular MTA on the Internet*, www.qmail.org

[05] *exim Internet Mailer*, www.exim.org

[06] *Procmail Homepage*, www.procmail.org

[07] *Java Apache Mail Enterprise Server*, http://jakarta.apache.org/james/

[08] *The Apache Software License, Version 1.1*,
http://jakarta.apache.org/james/license.html

[09] *JAVA Mail API*, http://java.sun.com/products/javamail/

[10] *Apache Jakarta Mailet API*, http://jakarta.apache.org/james/mailet/

[11] *JAVA Servlet Technology, the Power behind the Server*,
http://java.sun.com/products/servlet/

[12] *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*, 1997,
ftp://ftp.tu-dresden.de/pub/documents/rfc/rfc2104.txt

[13] *RFC 1321: The MD5 Message-Digest Algorithm*, 1992,
ftp://ftp.tu-dresden.de/pub/documents/rfc/rfc1321.txt

[14] *MySQL Documentation*, http://www.mysql.com/documentation/

[15] *A Plan for Spam*, 2002, http://www.paulgraham.com/spam.html

[16] *SpamGourmet*, www.spamgourmet.com

[17] *Teergrubing FAQ*, http://www.faqs.org/faqs/net-abuse-faq/teergrube-faq/

[18] *Tagged Message Delivery Agent (TMDA)-Homepage*, http://tmda.net/

[19] *The Postfix Homepage*, www.postfix.org

[20] *Pricing via Processing or Combatting Junk Mail*, 1992,
http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp.ps

[21] *HashCash*, 2002, http://www.cypherspace.org/~adam/hashcash/