Betrachtung von Datensicherheit in E-Mail Systemen und Entwicklung eines SMTP-basierten Anonymisierungs-Algorithmus

von Julian Ospald,

geboren am 07.08.1986 in Bielefeld

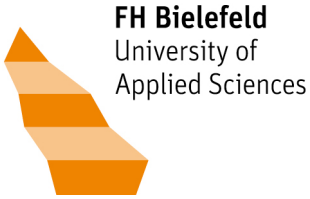
Betreut durch Frau Birgit Christina George

Bachelor-Abschlussarbeit,

eingereicht am Fachbereich Campus Minden

der FH Bielefeld, University of Applied Sciences

15. Juni 2016



Zusammenfassung

Im Angesicht globaler Überwachung durch Regierungen, schwer kontrollierbarer Sammlung und Speicherung von Benutzer-Daten durch Anbieter von Internetdiensten und fehlender Aufklärung über Konsequenzen internetbasierter Kommunikation soll diese Arbeit sowohl einen Überblick über aktuelle Probleme von E-Mail-Kommunikation, die die Privatsphäre des Benutzers betreffen, als auch eine Untersuchung möglicher Lösungen leisten.

Die Vielzahl möglicher Kommunikationsmittel im Internet erlaubt uns nicht, dieses Thema abstrakt zu behandeln. Oft unterscheiden sich diese fundamental: in der Art der Benutzung, dem verwendeten Protokoll, den zugrunde liegenden kryptografischen Algorithmen, den aktuellen Implementierungen und inhärenten Problemen.

Deshalb konzentriert sich diese Arbeit auf eines der ältesten digitalen Nachrichten-Übertragungsverfahren, welches schon im Arpanet erste Anwendung fand: die E-Mail. Dabei wird die E-Mail sowohl als Technologie aber auch als Kommunikationsform betrachtet und ihre Bedeutung aufgezeigt.

Der Überblick über aktuelle Probleme wird alle Aspekte behandeln, die relevant für die Privatsphäre des Benutzers sind. Dies beinhaltet konkrete Daten der Kommunikation wie Inhalt, aber auch Metadaten.

Bei der Betrachtung möglicher Lösungen werden zunächst bereits existierende aufgeführt, seien sie experimentell, schon implementiert oder nur konzeptionell. Nachfolgend wird ein alternativer Vorschlag inklusive Konzeption, rudimentärer Protokollbeschreibung, beinhaltender Algorithmen und Proof of Concept Implementierung der wissenschaftlichen Gemeinschaft zur Falsifikation unterbreitet.

Dieses Konzept soll als Idee, weniger als vollständige Lösung angesehen werden. Viele daraus folgende Probleme werden nicht ausreichend gelöst werden können. Diese werden jedoch nachfolgend untersucht und Vorschläge zur Weiterentwicklung unterbreitet.

Abschließend wird diskutiert, ob das vorgeschlagene Konzept ausreichend Potenzial hat, um bei entsprechender Weiterentwicklung eine angemessene Lösung sein zu können.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Kapitel soll eine Übersicht über die Bedeutung, Beschaffenheit und Funktionsweise der E-Mail geben, was sowohl Protokolle und Format als auch in der Praxis relevante Technologien betrifft. Ebenso beschreibt es das Zusammenwirken dieser Protokolle und das daraus entstehende Gesamtsystem, welches auch grob die in der Praxis aufzufindende Server-Struktur umreißt. Dies ist notwendig, um die Problematik praktischer Anonymisierung und Verschlüsselung nachfolgend darzulegen, welche sowohl technologischer als auch ökosystematischer Natur sind.

Die E-Mail ist eines der ältesten digitalen Nachrichten-Übertragungsverfahren und wurde bereits im Arpanet über Erweiterungen des FTP Protokolls (vgl. [55]) (vgl. [9]) angewendet. So gesehen entstand sie über einen längeren Zeitraum und entwickelte sich über mehrere RFCs über die Jahrzehnte. Streng genommen bezeichnet die E-Mail jede Form briefähnlicher Nachrichten, die auf elektronischem Wege übertragen werden.

Das am häufigsten benutzte E-Mail System setzt sich aus mehreren Komponenten zusammen, welche sehr stark miteinander korrelieren. Die Basis hierbei bildet das IMF vom RFC 5322 [41], welches nur die Form einer Nachricht spezifiziert. Darauf aufbauend wurde das SMTP Protokoll im RFC 5321 [26] entwickelt, welches den Transport von Nachrichten spezifiziert. Weiterhin existieren Protokolle, die die Verwaltung und den Zugriff auf E-Mails durch den Benutzer, nicht deren Transport zwischen Servern, beschreiben. Dazu zählen POP3 nach RFC 1939 [32] und IMAP nach RFC 3501 [8], die sich allerdings in ihrer Funktionalität teilweise stark unterscheiden.

Konzeptionell besteht die gängige E-Mail in erster Linie aus einem äußeren Envelope, welcher Meta-Daten für den Transport beinhaltet. In diesem befinden sich Header und Body. Der Header beinhaltet ebenfalls Metadaten und die eigentliche Nachricht wird als Body bezeichnet.

Dieses System bildet allerdings nur die minimale Basis heutiger internetbasierter E-Mail Kommunikation und wird in der Praxis durch eine Vielzahl von Technologien

erweitert. Manche davon wurden speziell als Erweiterungen für E-Mail Protokolle entwickelt, andere werden lediglich systemkompatibel eingesetzt.

Um die Relevanz heutiger E-Mail Kommunikation zu verstehen, ist es hilfreich Statistiken über Anzahl von E-Mail Konten und E-Mail Datenverkehr zu betrachten. Eine Studie der Radicati Group [43] im Jahr 2013 ergab, dass sich die Anzahl weltweiter E-Mail Konten auf ca. 3.9 Mrd. beläuft. Die Prognose für das Jahresende 2017 sah einen Anstieg auf ca. 4.9 Mrd. Konten vor, welches einer jährlichen Wachstumsrate von 6% entspricht. Ca. 24% aller E-Mail Konten sind dieser Studie zufolge geschäftliche E-Mail Konten. Der Datenverkehr insgesamt belief sich im Jahr 2013 auf ca. 182 Mrd. gesendeter E-Mails pro Tag. Es wird erwartet, dass dieses Volumen für geschäftliche E-Mails ansteigt. Zu bemerken ist allerdings, dass eine Abnahme des Volumens von Privatanwendern erwartet wird. Dies könnte laut der Radicati Group an dem Aufkommen neuer Kommunikationstechnologien wie Instant Messaging und sozialen Netzwerken liegen. Dennoch zeigen diese Zahlen, dass E-Mail Kommunikation äußerst relevant ist und sich als Kommunikationsform etabliert hat. (vgl. [43, S. 3, 4])

Aufgrund der hohen Verbreitung von E-Mail Kommunikation müssen auch Lösungen zur Anonymisierung diskutiert werden, die möglicherweise nicht optimal, aber mittelfristig praktikabel und kompatibel mit dem bestehenden System sind.

1.1 E-Mail Format (IMF)

Das im Kontext aktueller E-Mail Systeme benutzte E-Mail Format ist das »Internet Message Format«, kurz IMF.

Das IMF hat mehrere Versionen, angefangen beim RFC 822 [9], welches noch den Titel »STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES« trägt und vom 13. August 1982 ist, bis hin zu RFC 2822 [40], welches bereits den Titel »Internet Message Format« trägt und vom April 2001 ist. Auf Basis dieser beiden RFCs wurde im Oktober 2008 der RFC 5322 [41] veröffentlicht und ist damit die aktuellste Version zum Zeitpunkt dieser Arbeit. Das Ziel des RFC 5322 ist es eine Spezifikation für das Format von Text-Nachrichten, die zwischen Computer-Benutzern verschickt werden, bereitzustellen. Es ist im Kontext der E-Mail eingebettet und korreliert somit sehr eng mit dem SMTP Protokoll. Es beschreibt allerdings lediglich das Format der Nachrichten, nicht wie diese Nachrichten verschickt oder empfangen werden.

Das Format selbst setzt sich aus den Konzepten Header und Body zusammen. Der Header besteht aus mehreren Headerfeldern, welche jeweils einen Namen und einen Wert haben, die durch das Zeichen ":" getrennt werden. Die Headerfelder selbst werden durch einen Zeilenumbruch voneinander getrennt. Es gibt eine Reihe durch den RFC 5322 vordefinierter Headerfelder, die eine vorgegebene Bedeutung haben

und von E-Mail Servern in bestimmter Weise verwertet werden. Dazu zählen vor allem folgende:

- `From:` der Absender, beschrieben durch E-Mail Adresse, Name optional
- `To:` eine Liste von Empfängern, beschrieben durch E-Mail Adresse, Name optional
- `Date:` Zeitpunkt zu dem die Nachricht verfasst wurde, nicht Zeitpunkt des Transports
- `Subject:` Betreff der Nachricht
- `Message-ID:` eine von E-Mail Servern automatisch generierte ID wird intern benutzt, um mehrfache Zustellung zu verhindern
- `Received:` wird von E-Mail Servern, die nur weiterleiten, der Nachricht hinzugefügt, somit kann der Weg, den eine E-Mail nimmt, nachvollzogen werden

Die einzigen obligatorischen Headerfelder sind sind `From:` und `Date:`. Jeder Header, der diese Felder nicht besitzt, muss als nicht wohlgeformt zurückgewiesen werden. Weiterhin sind Headerfelder möglich, die nicht im RFC 5322 beschrieben sind. Diese können von E-Mail Servern ignoriert werden. Manche Headerfelder sind für E-Mail Clients gedacht oder werden zum Beispiel von Nachrichtenfiltern ausgewertet. Dem Header folgt nach einer leeren Zeile der Body. Dieser besteht aus der eigentlichen Nachricht. Sowohl Header als auch Body sind auf 7-bit ASCII beschränkt.

Eine vollständige E-Mail gemäß IMF könnte z.B. so aussehen:

```
1 From: Hans Bauer <hans@bauer.de>
2 To: Mina Meier <mina@meier.de>
3 Date: Tue, 8 Mar 2016 06:46:18 -0800
4 Subject: Beispiel-Mail
5 Message-ID: <18283.122131@bauer.de>
6 X-Mein-Header: True
7
8 Dies ist ein Beispieltext
9 über mehrere Zeilen.
```

Code 1.1: E-Mail

Zeile 1 bis 6 stellen den vollständigen Header dar, getrennt durch die Leere Zeile 7. Der Body erstreckt sich über die Zeilen 8 bis 9.

Aufgrund der Beschränkungen dieses Systems auf 7-bit ASCII Zeichen und dem Fehlen von Inhalten, die nicht Text sind, wurde »Multipurpose Internet Mail Extensions«, kurz MIME, entwickelt. Es ist beschrieben in RFC 2045 [17], RFC 2046 [18],

RFC 2047 [30], RFC 2049 [16], RFC 4288 [19] und RFC 4289 [20]. Die Integration dieses Systems in SMTP ist in RFC 1521 [5] und RFC 1522 [29] beschrieben.

MIME erweitert den IMF Standard, indem es Zeichen erlaubt, die außerhalb des 7-bit ASCII Bereichs liegen, sowohl für den Body als auch für die Header. Weiterhin ermöglicht es Inhalte, die nicht Text sind, wie Programme, Audio-Dateien, Video-Dateien oder Bilder. Anzumerken ist, dass MIME für unterschiedliche Protokolle verwendet wird, nicht nur für SMTP, sondern auch für beispielsweise HTTP (vgl. [4, Kapitel 4.7.10.3]).

Die Integration in den bestehenden IMF Standard erfolgt über neue dedizierte Headerfelder, die die nötigen Informationen beinhalten, um etwaige spezielle Headerfelder oder nicht-Text Teile des Bodies interpretieren zu können. Die relevanten Headerfelder lauten wie folgt:

- `MIME-Version:` die Version des MIME Standards
- `Content-Type:` der Typ des Inhalts, z.b. `text/plain`
- `Content-Disposition:` wurde in RFC 2183 [52] hinzugefügt und erlaubt das Angeben von sogenannten »Presentation Information«

Der MIME Standard erlaubt es aber vor allem sogenannte »MIME multipart messages« zu erstellen und zu interpretieren. Diese haben mehrere Bodies, die einzeln interpretiert werden. Getrennt werden diese durch das sogenannte »boundary«.

Eine E-Mail mit einem gewöhnlichen Datei-Anhang würde beispielsweise so aussehen:

```
1  From: Hans Bauer <hans@bauer.de>
2  To: Mina Meier <mina@meier.de>
3  Date: Tue, 8 Mar 2016 06:47:18 -0800
4  Subject: Beispiel-MIME-Mail
5  Message-ID: <18283.122131@bauer.de>
6  MIME-Version: 1.0
7  Content-Type: multipart/mixed;
8   boundary="------------080209010109060601000409"
9
10 This is a multi-part message in MIME format.
11 --------------080209010109060601000409
12 Content-Type: text/plain; charset=utf-8
13 Content-Transfer-Encoding: 7bit
14
15 Mit Anhang.
16
17 --------------080209010109060601000409
18 Content-Type: text/plain; charset=UTF-8;
19  name="Datei"
20 Content-Transfer-Encoding: base64
21 Content-Disposition: attachment;
22  filename="Beispieldatei"
23
24 RGF0ZWlpbmhhbHQK
25 --------------080209010109060601000409--
```

Code 1.2: MIME-Mail

Die ersten 8 Zeilen sind der E-mail Header, die 9. leere Zeile trennt Header und Body, welcher sich von Zeile 10 bis 25 erstreckt. Wichtig hierbei ist das Headerfeld `Content-Type:` in Zeile 7, welches ankündigt, dass dies eine multipart MIME E-Mail ist. Ebenso gibt dieses Feld in Zeile 8 über `boundary=` an, wo sich die einzelnen Teile im gesamten Body befinden. Dazu beinhalten die einzelnen Teile des Bodies erneut MIME Headerfelder, die eine korrekte Interpretation durch z.B. das E-Mail Programm vom Benutzer erlauben. So besitzt diese E-Mail einen gewöhnlichen Text-Anteil und einen Dateianhang, welcher Base64 verschlüsselt ist.

Dem Benutzer würde in diesem Fall im E-Mail Programm der Text `Mit Anhang.` angezeigt werden und die Möglichkeit angeboten werden, die Datei mit dem Namen `Beispieldatei` herunterzuladen.

1.2 E-Mail Protokolle

Dieses Unterkapitel behandelt die weitverbreiteten E-Mail Protokolle SMTP, POP3 und IMAP4, welche die Basis heutiger internetbasierter E-Mail Kommunikation bilden. Für diese Protokolle gibt es eine Vielzahl von Erweiterungen, die allerdings immer auf den bestehenden Protokollen aufbauen und diese selbst nicht verändern, damit die gemeinsame Schnittstelle nicht verloren geht.

Während das SMTP Protokoll die Kommunikation zwischen E-Mail Servern definiert, behandeln die Protokolle POP3 und IMAP4 die Verwaltung und den Zugriff auf E-Mails durch den Benutzer, nicht deren Transport zwischen Servern. Obwohl IMAP4 das modernere System ist, verwenden viele Server noch POP3, weshalb es hier der Vollständigkeit halber mit aufgeführt wird.

1.2.1 SMTP

Das SMTP Protokoll hat ähnlich wie das Internet Message Format mehrere Versionen, angefangen beim RFC 821 vom August 1982 [39] über RFC 2821 vom April 2001 [25] bis hin zu RFC 5321 vom Oktober 2008 [26], welches die aktuellste Version zum Zeitpunkt dieser Arbeit ist. Das Ziel des RFC 5321 ist es, eine Spezifikation für ein Protokoll zum Transport von E-Mails auf Basis des IMF bereitzustellen. Somit beschreibt es die grundlegende Kommunikation von E-Mail Servern untereinander, inklusive Versand, Empfang, Fehlerbehandlung etc., als auch die Kommunikation zwischen einem SMTP Client, welcher eine E-Mail einreicht, und einem SMTP/E-Mail Server. Besonders hervorzuheben ist auch, dass dieser RFC nicht nur das SMTP Protokoll an sich beschreibt, sondern auch den »*SMTP extension mechanism*« [26, S. 1, S. 9 ff] und diverse praktische Gesichtspunkte, die über die Protokoll-Spezifikation hinausgehen.

In seiner Grundform beschreibt das SMTP Protokoll ein Verfahren, das aus mehreren Schichten besteht. Die erste Schicht ist hierbei die Verbindung selbst, gewöhnlicherweise zwischen einem SMTP Client und einem SMTP Server (vgl. [26, S. 7]). Diese Verbindung wird generell über ein zuverlässiges Protokoll wie beispielsweise TCP hergestellt, da es keine Fehlertoleranz auf Daten-Ebene gibt. Sobald der Client sich mit dem Server verbunden hat, z.B. über das Programm »telnet«, stehen dem Client auf Verbindungsebene folgende Kommandos zur Verfügung:

- `HELO`: identifiziert den Client zum SMTP Server
- `QUIT`: beendet die Verbindung zum SMTP Server

Wie hier zu erkennen ist, ist SMTP ein textbasiertes Protokoll. Beim Senden von `HELO` muss der Client als Parameter seine FQDN mitschicken, erst danach ist die Verbindung erfolgreich zustande gekommen und der nächste Schritt ist möglich.

Hat die Verbindung diesen Zustand erreicht, so sind nachfolgend 0 oder mehr Transaktionen möglich, welches die nächste Schicht darstellt. Für eine einzelne Transaktion stehen dem Client folgende Kommandos zur Verfügung:

- `MAIL FROM`: der erste Schritt, initiiert eine Transaktion und gibt den Sender der E-Mail an
- `RCPT TO`: der zweite Schritt, gibt den Empfänger an
- `DATA`: der gewissermaßen letzte Schritt gibt an, dass jetzt der Inhalt der E-Mail gemäß dem IMF folgt

Die weiteren Informationen auf Transaktions-Ebene wie `MAIL FROM` und `RCPT TO` in Verbindung mit der tatsächlichen E-Mail, die über das `DATA` Kommando übermittelt wird, können auch grob als Envelope betrachtet werden.

Anzumerken ist hier, dass der E-Mail Server die Transaktion abbrechen muss, wenn diese Kommandos in fehlerhafter Reihenfolge ankommen. Die Reihenfolge ist genau einzuhalten, d.h. erst `MAIL`, welches einmalig angegeben wird, dann `RCPT`, welches mehrmals angegeben werden kann, dann `DATA` und anschließend der Inhalt. Wenn der Inhalt der E-Mail gemäß IMF inklusive Header gesendet wird, so muss dieser mit einer Zeile enden, die nur einen Punkt enthält (vgl. [26, S. 20]).

Nach jedem Kommando, das der Client sendet, muss der Server antworten. Dies geschieht über den Status-Code (engl. »*reply code*« [26, S. 48 ff]), welcher aus 3 Ziffern besteht. Die erste Ziffer gibt an, ob die Antwort gut, schlecht oder unvollständig ist. Die zweite Ziffer gibt an, welche Art von Fehler vorliegt. Die dritte Ziffer gibt weitere Informationen über den Fehler an. Eine vollständige Betrachtung der Status-Codes macht an dieser Stelle keinen Sinn, deshalb wird hier nur die erste Ziffer betrachtet, um einen groben Überblick zu verschaffen. `2xx` bedeutet, dass die Anfrage erfolgreich verarbeitet wurde und eine weitere Anfrage gesendet werden kann. `3xx` bedeutet ähnlich wie `2xx`, dass die Anfrage erfolgreich war, allerdings erwartet der Server jetzt noch weitere Informationen vom Client. Dies tritt beispielsweise beim `DATA` Kommando auf, nach welchem der E-Mail Inhalt folgen muss. `4xx` denotiert ein Kommando, das nicht akzeptiert wurde, was bedeutet, dass die Anfrage nicht ausgeführt wurde. Der Client kann bzw. sollte die Kommando-Sequenz wiederholen. `5xx` denotiert ähnlich wie `4xx` eine abgewiesene Anfrage. Allerdings ist dies ein permanenter Fehler und die zugrunde liegende Kommando-Sequenz sollte nicht wiederholt werden. Nach dem Status-Code folgt immer ein Text String, der menschenlesbar sein sollte. (vgl. [26, S. 47 ff]).

Eine vollständige SMTP Sitzung könnte wie folgt aussehen (C steht für Client, S für Server):

```
1  C: telnet mail.wurst.de 25
2  S: 220 service ready
3  C: HELO wurst.de
4  S: 250 OK
5  C: MAIL FROM:<hans@bauer.de>
6  S: 250 OK
7  C: RCPT TO:<mina@meier.de>
8  S: 250 OK
9  C: RCPT TO:<wilhelm@keiser.de>
10 S: 250 OK
11 C: DATA
12 S: 354 start mail input
13 C: From: Hans Bauer <hans@bauer.de>
14    To: Mina Meier <mina@meier.de>,
15        Kaiser Wilhelm <wilhelm@keiser.de>
16    Date: Tue, 8 Mar 2016 06:46:18 -0800
17    Subject: Beispiel-Mail
18    Message-ID: <18283.122131@bauer.de>
19    X-Mein-Header: True
20
21    Dies ist ein Beispieltext
22    über mehrere Zeilen.
23    .
24 S: 250 OK
25 C: QUIT
26 S: 221 closing channel
```

Code 1.3: SMTP-Sitzung

Was hier besonders auffällt und äußerst wichtig für die weitere Betrachtung der Problematik und Implementierung sein wird, ist die Tatsache, dass die Headerfelder `From` und `To` auf Ebene des IMF praktisch losgelöst sind von den tatsächlich angegebenen Informationen `MAIL FROM` und `RCPT TO` auf Ebene des SMTP Protokolls. Sie können divergieren oder übereinstimmen. Obwohl es hier eine Reihe von Regeln gibt wie sich ein E-Mail Server verhalten soll, gibt es keine Garantie, dass hier Konsistenz herrscht. Dies wird unter anderem für das Headerfeld `BCC` genutzt, welches angibt, dass die E-Mail an einen weiteren Empfänger gesendet werden soll, aber diese Information aus dem Header gelöscht wird, sobald der Server zustellt oder weiterleitet (vgl. [26, S. 85]).

Ebenso ersichtlich in Zeile 1 ist hier, dass eine SMTP Sitzung generell über Port 25 abgewickelt wird (vgl. [26, S. 68]). Reicht ein SMTP Client eine E-Mail bei einem

Server ein, wird allerdings Port 587 gemäß dem RFC »*Message Submission for Mail*« [21, S. 6] benutzt. Im Falle von SSL/TLS Verschlüsselung wird allerdings häufig Port 465 benutzt (vgl. [1]), obwohl nach RFC »*SMTP Service Extension for Secure SMTP over TLS*« [23, S. 2] dafür kein designierter Port mehr vorhergesehen ist, da stattdessen STARTTLS benutzt werden soll.

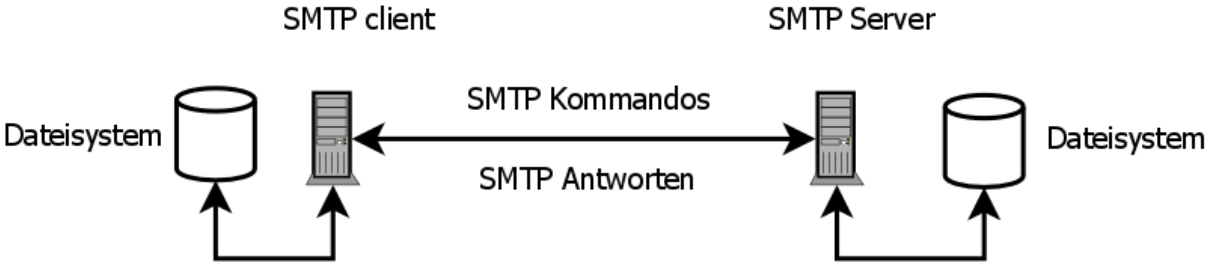


Abbildung 1.1: SMTP Modell

1.2.2 POP3

POP3 ist die Version 3 des »Post Office Protocol«, spezifiziert in RFC 1939 [32]. Während SMTP das Versenden, Erstellen und Weiterleiten von E-Mails beschreibt, beschreibt POP3 das Abholen, Auflisten und Löschen von E-Mails an einem E-Mail Server (vgl. [32, S. 2]). Es ist keine Erweiterung des SMTP Protokolls, da es eine eigenständige Funktionalität hat, die SMTP nicht bereitstellt. Ebenso denkbar ist, dass POP3 mit einem E-Mail Server verwendet wird, der nicht SMTP zum Versenden benutzt, was allerdings in der Praxis kaum vorkommt. POP3 ist sehr simpel und eingeschränkt in der Funktionalität, was bereits in RFC 1939 angesprochen wird, mit einem Vermerk zu dem etwas mächtigeren IMAP4 Protokoll (vgl. [32, S. 2]).

Ähnlich wie SMTP ist auch POP3 ein Client-Server basiertes System. In gängigen E-Mail Servern befindet sich also häufig nicht nur ein SMTP, sondern auch ein POP3 Server.

Ebenso wie SMTP ist auch POP3 TCP basiert. Die Kommunikation findet entweder über Port 110 (vgl. [32, S. 3]) statt oder im Falle von SSL/TSL Verschlüsselung über das POP3S Protokoll auf Port 995, welcher allerdings nach RFC 2595 (vgl. [34, S. 9]) als veraltet gilt und stattdessen STARTTLS auf dem Standardport 110 benutzt werden soll.

POP3 ist ein sehr zustandsbehaftetes Protokoll, d.h. die Verbindung kann sich in verschiedenen Zuständen befinden. Der erste Zustand ist der »*AUTHORIZATION state*« [32, S. 3], in dem sich die Verbindung befindet nachdem der POP3 server das sogenannte »*greeting*« [32, S. 3] gesendet hat. Nachfolgend muss sich der Client authentifizieren. Ist die Authentifizierung erfolgreich, so gelangt die Verbindung in den »*TRANSACTION state*« [32, S. 3]. In diesem Zustand kann der Client diverse

Anfragen senden, um mit den auf dem POP3 Server liegenden E-Mails zu interagieren. Der letzte Zustand, der sogenannte »*UPDATE state*« [32, S. 3 f], wird erreicht nachdem der Client das `QUIT` Kommando gesendet hat. In diesem Zustand werden alle Ressourcen freigegeben, die während des »TRANSACTION state« belegt wurden. Erst danach wird die TCP Verbindung geschlossen. (vgl. [32, S. 3 f])

Dem POP3 Client stehen folgende Kommandos zur Kommunikation mit dem Server zur Verfügung:

- AUTHORIZATION state
  - `USER <user>`: wird in Verbindung mit `PASS` zur Authentifizierung verwendet, alternativ kann `APOP` verwendet werden
  - `PASS <password>`: wird in Verbindung mit `USER` zur Authentifizierung verwendet, alternativ kann `APOP` verwendet werden
  - `APOP`: alternatives Authentifizierungsverfahren, welches das Passwort nicht als Klartext sendet
  - `QUIT`: beendet die Verbindung
- TRANSACTION state
  - `STAT`: Abfrage des Status der Mailbox
  - `LIST (i)`: zeigt Informationen zu der i-ten E-Mail oder zu allen E-Mails, gewöhnlicherweise Anzahl und Größe
  - `RETR i`: lädt die i-te E-Mail vom Server herunter
  - `DELE i`: löscht die i-te E-Mail
  - `NOOP`: tut nichts, der Server antwortet immer positiv
  - `RSET`: macht das löschen von E-Mails rückgängig
  - `TOP i l`: lädt den Header und die ersten l Zeilen der i-ten E-Mail herunter
  - `UIDL i`: zeigt die »unique-id« der E-Mail an
- UPDATE state
  - `QUIT`: beendet die Verbindung und führt mögliche `DELE` Kommandos aus

Eine mögliche POP3-Sitzung könnte so aussehen:

```
1  S: <wartet auf Verbindung an TCP port 110>
2  C: <client verbindet sich>
3  S: +OK pop3.bauer.de POP3-Server
4  C: USER hans@pop3.bauer.de
5  S: +OK Valid user, now enter password
6  C: PASS <Klartext-Passwort>
7  S: +OK Password valid, maildrop locked and ready
8  C: STAT
9  S: +OK 500 4670
10 C: LIST 670
11 S: -ERR no such message
12 C: LIST 67
13 S: +OK 67 300
14 C: RETR 67
15 S: +OK 300 octets
16 S: <der POP3 server sendet die vollständige E-Mail>
17 S: .
18 C: DELE 67
19 S: +OK message 67 deleted
20 C: QUIT
21 S: +OK POP3 server signing off (maildrop empty)
22 C: <beendet Verbindung>
23 S: <wartet auf Verbindung an TCP port 110>
```

Code 1.4: POP3-Sitzung

Wie hier zu sehen ist, liegt ein impliziter Zustand vor, welcher aus dem Kontext erschlossen werden muss.

Wichtig anzumerken ist auch, dass das normale POP3 Authentifizierungsverfahren über die Kommandos `USER` und `PASS` unsicher ist, da die Daten als Klartext gesendet werden (vgl. [32, S. 14]). Ebenso gilt das alternative APOP Authentifizierungsverfahren als unsicher (vgl. [44, S. 1-18]). Stattdessen kann STARTTLS mit POP3 verwendet werden, gemäß RFC 2595 [34].

1.2.3 IMAP4

IMAP4 ist die Version 4 des »INTERNET MESSAGE ACCESS PROTOCOL« spezifiziert in RFC 3501 [8]. Es ist ähnlich wie POP3 ein textbasiertes Client-Server Protokoll, das das Auflisten, Löschen und Manipulieren von E-Mails an einem E-Mail Server erlaubt, weit über die Möglichkeiten von POP3 hinaus (vgl. [32, S. 2]).

Ein E-Mail Server kann sowohl POP3 als auch IMAP4 gleichzeitig anbieten. Im Gegensatz zu POP3 erlaubt IMAP4 allerdings nicht nur E-Mails herunterzuladen, zu löschen oder aufzulisten, sondern direkt auf der Mailbox des E-Mail Servers zu operieren, was Funktionalitäten wie neue Ordner anzulegen erlaubt (vgl. [8, S. 34 f]) oder das serverseitige Durchsuchen von Nachrichten (vgl. [8, S. 49 ff]).

Informationen werden nach Bedarf direkt vom IMAP4 Server abgerufen und nicht einzeln heruntergeladen. Deshalb ist auch im Gegensatz zu POP3 eine lokale Speicherung von Daten nicht notwendig, welche auch dazu führen kann, dass die lokale Mailbox von der entfernten Mailbox divergiert. E-Mail Programme bieten jedoch unabhängig von POP3/IMAP4 die Möglichkeit der lokalen Speicherung an, was allerdings Protokoll-agnostisch ist. Bei IMAP4 ist anzumerken, dass bei fehlender Internetverbindung und keiner expliziten lokalen Speicherung auf Programmebene der Zugriff auf E-Mails auch nicht mehr möglich ist.

IMAP4 benötigt ein zuverlässiges Netzwerkprotokoll wie beispielsweise TCP und wird über den Port 143 abgewickelt (vgl. [8, S. 6]). Ist die Verbindung über IMAPS TLS/SSL-verschlüsselt wird häufig Port 993 benutzt, obwohl in RFC 2595 [34] STARTTLS über den Standardport 143 beworben wird.

Ebenso wie POP3 ist auch IMAP4 ein zustandsbehaftetes Protokoll. Allerdings ist IMAP4 weitaus komplexer und kann sich in 4 verschiedenen Zuständen befinden. Wenn die Verbindung vom Client zum Server initial erstellt wurde, befindet sich die Kommunikation im »*Not Authenticated State*« [8, S. 13]. Es wird vom Client nun erwartet, sich zu authentifizieren. Findet eine erfolgreiche Authentifizierung statt, so wird der »*Authenticated State*« [8, S. 13] erreicht. Nachfolgend muss der Client eine Mailbox auswählen, bevor er weitere Kommandos zur Manipulation oder Einsicht von E-Mails abgeben kann. Ebenso wird dieser Zustand erreicht, wenn es einen Fehler beim Auswählen einer Mailbox gab oder nach einem erfolgreichen `CLOSE` Kommando (vgl. [8, S. 13]). Wurde eine Mailbox erfolgreich ausgewählt, so erreicht die Verbindung den »*Selected State*« [8, S. 13]. In diesem Zustand kann auf der Mailbox operiert werden wie z.B. Ordner anlegen, E-Mails abrufen etc. Der letzte Zustand ist der »*Logout State*« [8, S. 14] und kann aus allen vorherigen Zuständen erreicht werden. Hier wird die Verbindung terminiert.

Eine vereinfachte Darstellung der Kommunikation in einem SDL Diagramm ist in Abbildung 1.2 zu sehen. Diese Abbildung zeigt bereits die Komplexität des Protokolls, ist aber nicht vollständig. Deshalb kann an dieser Stelle aufgrund der Vielzahl möglicher Kommandos keine vollständige Betrachtung vorgenommen werden. Stattdessen wird nachfolgend eine einfache IMAP4 Sitzung betrachtet:

```
C: <öffnet Verbindung>
S: * OK IMAP4rev1 Service Ready
C: a001 login mrc secret
S: a001 OK LOGIN completed
C: a002 select inbox
S: * 322 EXISTS
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * 12 RECENT
S: * OK [UNSEEN 88] Message 88 is the first unseen message
S: a002 OK [READ-WRITE] SELECT completed
C: a003 fetch 8 body[header]
S: * 8 FETCH (BODY[HEADER] {342}
S: From: Hans Bauer <hans@bauer.de>
S: To: Mina Meier <mina@meier.de>,
S:     Kaiser Wilhelm <wilhelm@keiser.de>
S: Date: Tue, 8 Mar 2016 06:46:18 -0800
S: Subject: Beispiel-Mail
S: Message-ID: <18283.122131@bauer.de>
S: X-Mein-Header: True
S:
S: )
S: a003 OK FETCH completed
C: a004 store 8 +flags \deleted
S: * 8 FETCH (FLAGS (\Seen \Deleted))
S: a004 OK +FLAGS completed
C: a005 logout
S: * BYE IMAP4rev1 server terminating connection
S: a005 OK LOGOUT completed
```

Code 1.5: IMAP4-Sitzung

Wie hier zu sehen ist, sind die Transaktionen durchnummeriert und haben für Client und Server denselben Bezeichner. Der IMAP4 Zustand ist implizit und muss aus dem Kontext erschlossen werden.

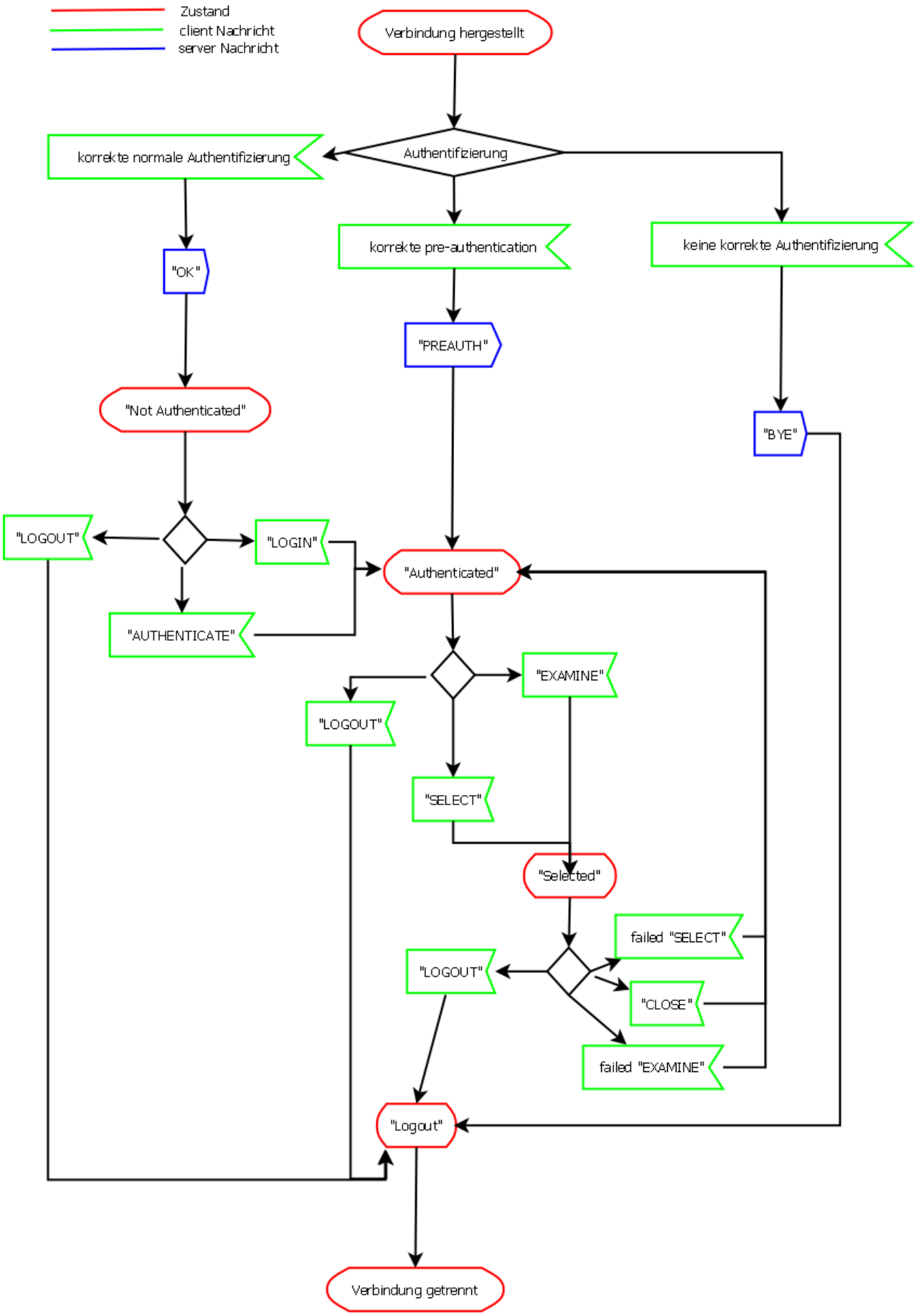


Abbildung 1.2: IMAP4 SDL Diagramm

1.3 E-Mail als Gesamtsystem

Nachdem nun die Spezifikationen und Protokolle im einzelnen vorgestellt wurden, muss die E-Mail als Gesamtsystem betrachtet werden. Dazu wird nachfolgend untersucht, wie die Komponenten zusammengeführt werden, miteinander kommunizieren und wie eine Server-Konfiguration grob in der Praxis aussieht.

Auf der untersten Ebene befindet sich der »Mail User Agent«, kurz MUA. Er bezeichnet ein gewöhnliches E-Mail Programm, mit dem E-Mails gelesen, verfasst, empfangen und gesendet werden können. Beispiele hierfür sind Programme wie etwa »Mozilla Thunderbird« oder »Outlook Express«. Dies ist die Stelle, an der der Benutzer dem System Input gibt. Ein solches Programm muss in der Lage sein, mit allen Protokollen und Spezifikationen umzugehen. Für den Versand ist das beispielsweise IMF und SMTP, für das Empfangen und Manipulieren von E-Mails beispielsweise POP3 oder IMAP4. Dies findet alles beim Benutzer statt, ist also clientseitig.

Auf der nächsten Ebene befindet sich der sogenannte E-Mail Server. Allerdings besteht dieser aus mehreren Komponenten. Mehrere E-Mail Server können sehr unterschiedliche Funktionalitäten bereitstellen, obwohl sie alle das SMTP Protokoll implementiert haben. Wenn der Benutzer über einen MUA eine E-Mail verschicken will, so sendet er diese an einen sogenannten »Mail Submission Agent« häufig an Port 587 (vgl. [21, S. 6]), kurz MSA. Über diesen wird die initial erstellte E-Mail empfangen und gelangt so in den Zustellungsmechanismus.

Nachfolgend kann die E-Mail nun, abhängig vom Zustellungsmechanismus, über mehrere sogenannte »Mail Transfer Agents«, kurz MTA, weitergeleitet werden. Für jede Weiterleitung wird der E-Mail ein Headerfeld `Received:` hinzugefügt, über welches der Weg, den die E-Mail über mehrere MTAs genommen hat, nachvollzogen werden kann. Der MTA ist die zentrale Einheit der E-Mail Zustellung, da er E-Mails versendet und empfängt.

Gelangt die E-Mail zum E-Mail Server des Empfängers, so wird der sogenannte »Mail Delivery Agent«, kurz MDA, aktiv. Dieser hat die Aufgabe, die E-Mail dem korrekten Benutzerkonto zuzuordnen, da ein E-Mail Server für gewöhnlich mehrere Benutzerkonten bereitstellt.

Auf der letzten Ebene befindet sich erneut ein MUA, und zwar in diesem Fall der des Empfängers, der nun über einen in seinem E-Mail Server integrierten POP3 oder IMAP4 Dienst die E-Mail abrufen kann.

Das bedeutet, dass ein E-Mail Server verschiedene Formen haben kann, z.B.:

- ein MTA, ein MSA, ein MDA und ein IMAP4 und POP3 Server, gewissermaßen als vollständige Lösung
- nur ein MSA und MTA, der empfangene E-Mails an einen vordefinierten E-Mail Server weiterleitet, gewissermaßen als SMTP Gateway

- nur ein MTA, z.B. als offenes oder internes Relay

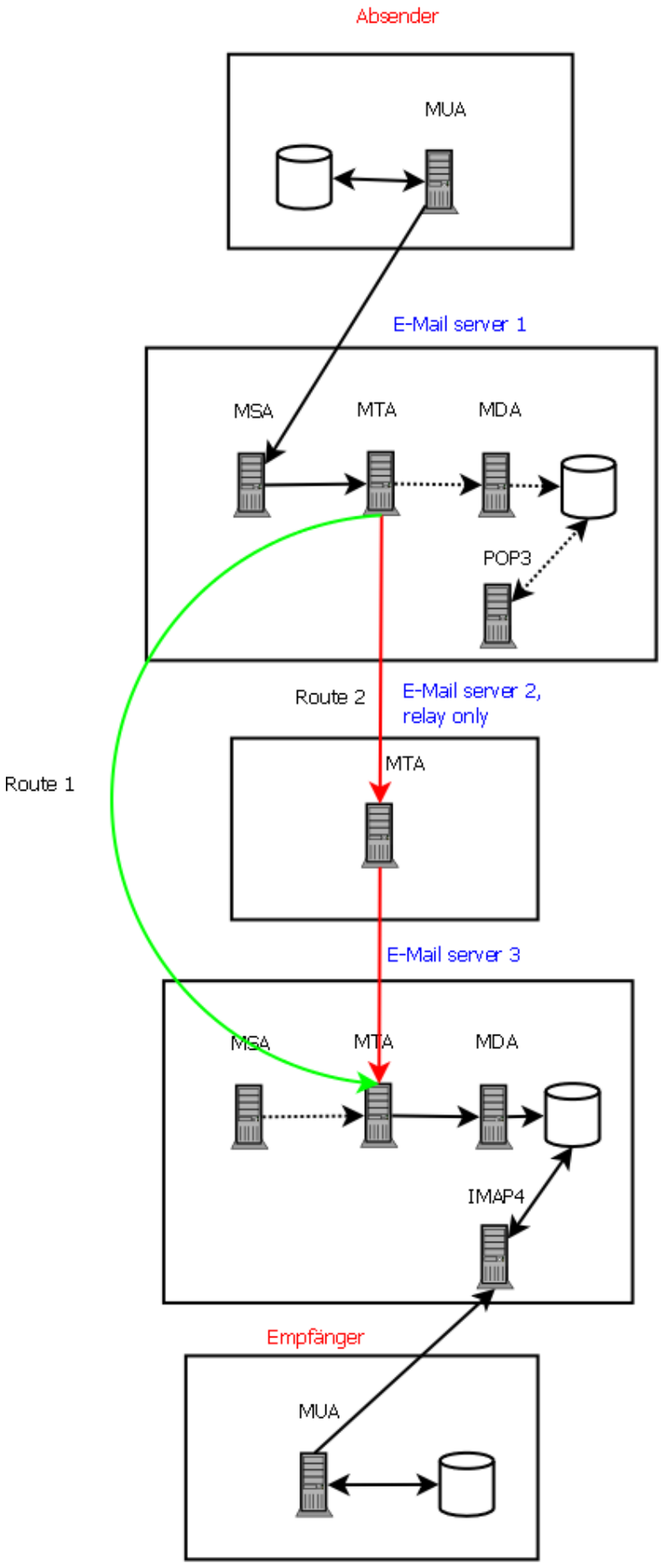


Abbildung 1.3: E-Mail als Gesamtsystem

Die Abbildung 1.3 zeigt schematisch das Zusammenspiel der Komponenten: Der ursprüngliche Absender erstellt über seinen MUA die E-Mail. Der MUA kontaktiert den MSA des »E-Mail Server 1« des Absenders, welcher im MUA hinterlegt ist. Der MTA des »E-Mail Server 1« muss jetzt entscheiden, was mit der E-Mail im nachfolgenden passiert. Eine Möglichkeit ist, dass die E-Mail direkt an den E-Mail Server des Empfängers gesendet wird (in Abbildung 1.3 als »Route 1« gekennzeichnet). Ebenso möglich ist aber auch, dass der Empfänger E-Mail Server ein MTA ist, der lediglich weiterleitet (in Abbildung 1.3 als »Route 2« gekennzeichnet). Die Regeln dazu werden in Routingtabelle und Zugriffsrichtlinien in den jeweiligen MTAs hinterlegt und können abhängig von einer Reihe von Informationen sein wie z.B. Empfängerdomain der E-Mail. Es kann notwendig sein, dass der weiterleitende MTA das Envelope der E-Mail manipuliert und z.B. den Empfänger anpasst. Wenn keine Regeln vorhanden sind, spricht man von einem »open relay«. Erreicht die E-Mail den »E-Mail Server 3« des schlussendlichen Empfängers, so wird dieser feststellen, dass er der Empfänger ist und leitet die E-Mail an den MDA weiter, der sie dem korrekten Benutzerkonto zustellt. Dieses Benutzerkonto ist über einen IMAP4 Server vom MUA des Empfängers erreichbar.

Es ist nicht notwendig, dass E-Mail Server 1, 2 und 3 echte physische Instanzen sind. Es sind auch komplexere Konfigurationen möglich, bei denen die einzelnen Funktionalitäten über mehrere physische Instanzen verteilt sind.

Das Konzept eines E-Mail Servers, der nur die Funktionalität eines MTA bereitstellt, wird nachfolgend noch von hoher Relevanz sein.

Ein weiteres wichtiges Detail für die Zustellung und Weiterleitung von E-Mails ist der sogenannte MX Resource Record, erstmals beschrieben in RFC 947 [37]. Beim Versenden oder Weiterleiten von E-Mails fragt der sendende MTA zunächst den MX-Record der Empfängerdomain ab, welcher die tatsächliche FQDN des E-Mail Servers des Empfängers liefert. Erst zu diesem verbindet er sich dann.

Kapitel 2

Motivation und Problematik

Wie durch Kapitel 1 erhellt wurde, sind die involvierten Protokolle für E-Mail Kommunikation äußerst verbose und übermitteln eine Menge an Metadaten und Inhaltsdaten. Standardmäßig werden alle diese Daten als Klartext übermittelt. Dazu gehören: Absender, Empfänger, Passwörter, Inhalt der E-Mail, etc. Dies bezieht sich auf nahezu alle Stationen, die eine E-Mail über das Internet durchläuft, mindestens aber folgende:

- MUA kommuniziert mit entfernten MSA
- MTA kommuniziert mit entfernten MTA
- MUA kommuniziert mit entfernten POP3/IMAP4 Server

Bei komplexeren Konfigurationen kann es sogar möglich sein, dass Informationen innerhalb einer Server-Konfiguration (z.B. entfernte Datenbank) sichtbar werden.

Um die Privatsphäre des Benutzers zu schützen, müssen so wenig Daten wie möglich und nur so viel wie nötig versendet werden. Die Daten, die gesendet werden müssen, sollten durch ein adäquates kryptografisches Verfahren verschlüsselt sein. Dies bezieht sich sowohl auf Metadaten als auch auf den Inhalt der E-Mail.

Nahezu alle existierenden Protokolle wie SMTP, IMAP4 und POP3 besitzen eine gewisse Unterstützung für kryptografische Methoden, die nachfolgend noch dargelegt werden. Allerdings können aufgrund der Beschaffenheit der genannten Protokolle immer noch Verbindungsdaten abgefangen werden, selbst wenn die Transaktionen (z.B. die SMTP Sitzung) kryptografisch verschlüsselt sind. Keines der existierenden E-Mail Protokolle bietet für letzteres eine adäquate Lösung. Dies ist nämlich das Problem der Anonymisierung von Metadaten, die nicht durch Verschlüsselung selbst versteckt werden können.

Anonymisierung von Metadaten in diesem Kontext unterscheidet sich also von der kryptografischen Verschlüsselung dadurch, dass, anstelle von Daten wie dem Inhalt einer Nachricht oder Transaktion, die Existenz der Transaktion oder Verbindung selbst für Außenstehende nicht nachvollziehbar ist, also die Verschleierung der Metadaten auf

Netzwerkebene. Dies wird häufig über dedizierte Systeme erreicht, die über Routing-Algorithmen die Indirektion einer Verbindung vervielfachen. Diese Indirektion führt dazu, dass zum Nachvollziehen der Verbindung alle Stationen der Route bekannt sein müssen. Ein Beispiel eines solchen Algorithmus ist das »*Onion Routing*« [47].

Allerdings sind alle genannten Protokolle in der Lage, zumindest auf der Seite der kryptografischen Verschlüsselung etwas zu leisten.

Demnach liegt dieser Arbeit die Motivation zugrunde, das E-Mail System vor allem hinsichtlich der Anonymität für den Benutzer zu verbessern. Ob bestehende Systeme dies leisten können oder ob völlig neue Systeme entwickelt werden müssen, soll hier diskutiert werden. In dem Zuge wird nachfolgend der aktuelle Stand der Verschlüsselung und Anonymität von E-Mail Kommunikation untersucht und die Anforderungen an eine adäquate Lösung formuliert.

2.1 Betrachtung von Datenschutz in aktuellen E-Mail Systemen

Dieser Abschnitt betrachtet den Datenschutz von aktuellen SMTP-basierten E-Mail Systemen hinsichtlich der Verschlüsselung und Anonymität.

2.1.1 Verschlüsselung von E-Mail-Verkehr

Mit Verschlüsselung sind kryptografisch-mathematische Methoden gemeint, um Nachrichten (z.B. Text) in eine neue Form zu bringen, die nicht ohne weiteres (z.B. ohne Schlüssel) in die ursprüngliche Form gebracht werden kann.

Bei der Betrachtung der Verschlüsselung von E-Mail Verkehr muss erst definiert werden welche Teile des Verkehrs verschlüsselt werden. Hier gibt es mehrere Schichten. Die erste Schicht sind die Inhaltsdaten der E-Mail, gewissermaßen der Body und eventuell auch die Header. Die zweite Schicht ist die Sitzung zwischen E-Mail Server und Client. Dies gilt für alle involvierten Protokolle SMTP, POP3 und IMAP4, die diese Verschlüsselung unterstützen müssen, um die einzelnen Kommandos vom Client zum Server und die Antworten vom Server zum Client über einen kryptografischen Kanal zu transferieren.

Zunächst werden Technologien bezüglich der Verschlüsselung des E-Mail Bodies betrachtet und nachfolgend Technologien auf Protokoll-Ebene untersucht. Dabei werden auch Probleme dieser Verfahren und Methoden aufgezeigt.

GPG

»*GNU Privacy Guard*« [48] ist eine vollständige und freie Implementierung des OpenPGP Standards, der in RFC 4880 [6] definiert ist.

Es handelt hier sich um ein Verschlüsselungsverfahren, das asymmetrische Public-Key Verfahren mit symmetrischen Verfahren kombiniert (vgl. [6, S. 1]).

Ein symmetrisches Verschlüsselungsverfahren benutzt nur einen einzigen Schlüssel für das Verschlüsseln und Entschlüsseln. Dieser muss also beiden Kommunikationspartnern zur Verfügung stehen, darf aber niemand anderem zugänglich sein. Die Übertragung eines solchen Schlüssels wird deshalb häufig über ein asymmetrisches Verschlüsselungsverfahren bewerkstelligt.

Ein asymmetrisches Public-Key Verfahren besteht auf Sender -und Empfänger-Seite jeweils aus einem Schlüsselpaar. Dieses setzt sich aus einem öffentlichen und einem privaten Schlüssel zusammen, die gemeinsam generiert werden. Nur die öffentlichen Schlüssel werden ausgetauscht und erlauben der jeweils anderen Partei für den Besitzer des Schlüsselpaars zu verschlüsseln. Der private Schlüssel wird vom Besitzer zum Entschlüsseln benutzt und kann mit einem Passwort geschützt sein.

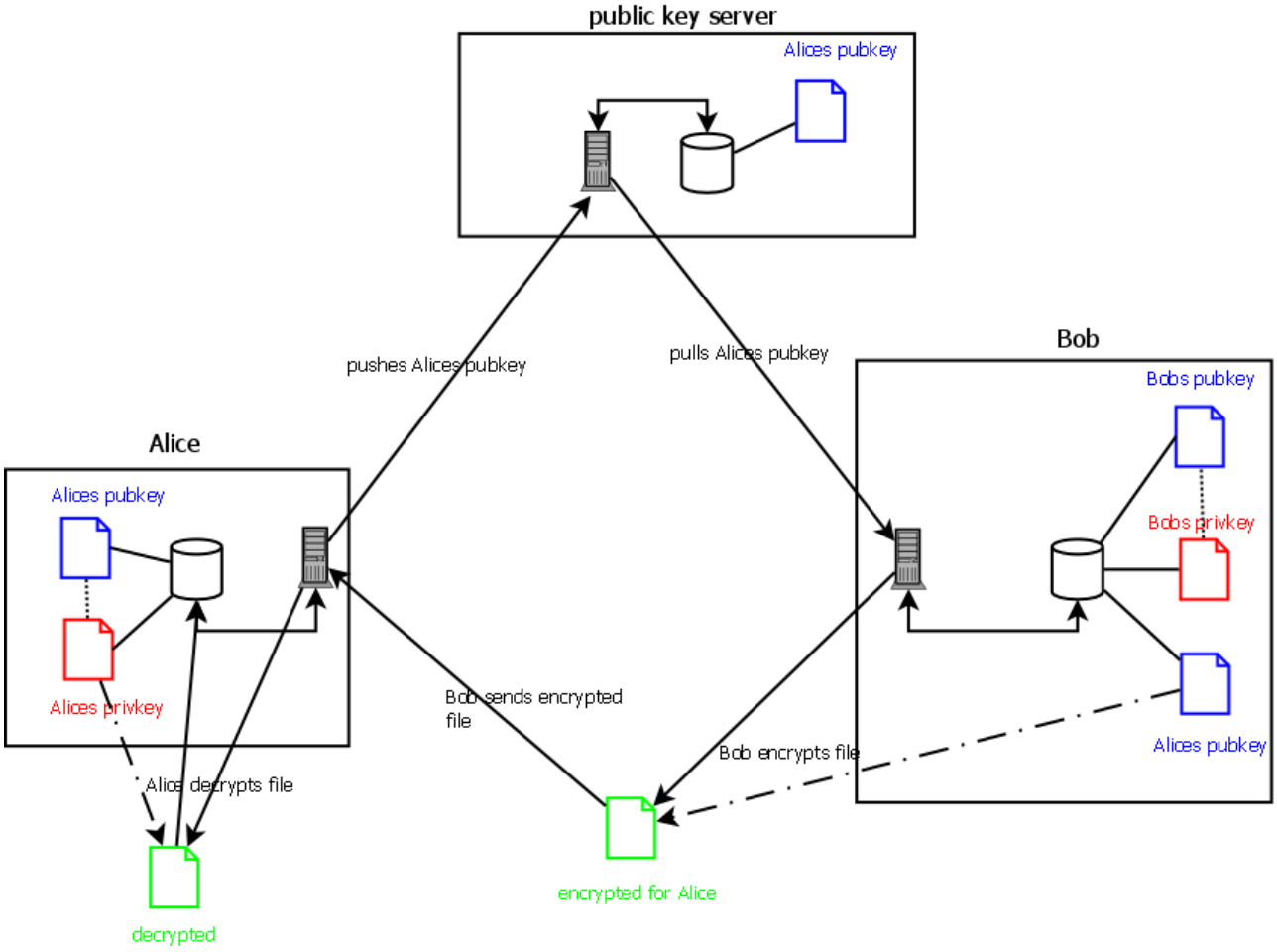


Abbildung 2.1: Asymmetrisches Verschlüsselungsverfahren in der Praxis

Abbildung 2.1 stellt schematisch einen vereinfachten Ablauf einer verschlüsselten einseitigen Kommunikation von Bob zu Alice dar, basierend auf dem asymmetrischen Verschlüsselungsverfahren. Beide Parteien benötigen zunächst ein Schlüsselpaar, welches lokal generiert wird. Danach müssen die Parteien die benötigten öffentlichen Schlüssel austauschen. Dies kann manuell oder über einen öffentlichen Server geschehen. Als nächsten Schritt benutzt Bob den öffentlichen Schlüssel von Alice, um die Nachricht für Alice zu verschlüsseln. Dann verschlüsselt und sendet er diese. Anschließend benutzt Alice ihren privaten Schlüssel um die Nachricht zu entschlüsseln. (vgl. [15, Kapitel 3.3])

Hierbei kommt die Frage auf, woher Bob weiß, dass er den korrekten öffentlichen Schlüssel von Alice besitzt, wenn er diesen über einen öffentlichen Server geladen hat. Um dies festzustellen, wird ein »Secure Hash Algorithm« benutzt, der gewissermaßen den Fingerabdruck des öffentlichen Schlüssels darstellt. Bob kann nun Alice über einen sicheren Kanal kontaktieren (z.B. persönlich) und fragen, ob ihr Fingerabdruck des öffentlichen Schlüssels dem entspricht, den er lokal sieht. Aufgrund der Kollisionsunwahrscheinlichkeit von z.B. SHA-1 erlaubt dies eine sinnvolle Aussage über die Echtheit des öffentlichen Schlüssels, den Bob heruntergeladen hat. (vgl. [15, Kapitel 6.2.2])

GPG ist im Ablauf also ähnlich wie Abbildung 2.1, benutzt allerdings auch das symmetrische Verschlüsselungsverfahren in dem Verschlüsselungsprozess. Beim Sender wird dabei die Nachricht selbst lediglich über das symmetrische Verfahren mit einem einmaligen »session key« verschlüsselt. Dieser »session key« wird dann mit dem öffentlichen Schlüssel des Empfängers über das asymmetrische Verschlüsselungsverfahren verschlüsselt. Der symmetrische Schlüssel wird dann am Anfang der Nachricht mitgeschickt. Der Empfänger muss dann erst mit seinem privaten Schlüssel den »session key« entschlüsseln und dann mit diesem die eigentliche Nachricht. Dies geschieht natürlich auf Programmebene und wird nicht manuell vom Benutzer durchgeführt. (vgl. [6, S. 6])

Anzumerken ist, dass GPG nicht nur für Verschlüsselung, sondern auch für Signierung benutzt werden kann. Damit kann der Absender einer Nachricht kryptografisch verifiziert werden. (vgl. [6, S. 7])

GPG wird in der Praxis vielfältig benutzt. Dazu gehört das manuelle Verschlüsseln von Dateien für Andere, das Signieren von »git commits« und das Verschlüsseln des Nachrichteninhaltes einer E-Mail.

Angenommen, die E-Mail selber ist über das SMTP Protokoll in sicherer Weise weitergeleitet worden, ohne dass ein Dritter diese verändern oder sehen konnte. Wenn die E-Mail auf dem Zielserver eintrifft, liegt sie dort dennoch als Klartext vor und wird durch die verschiedenen Komponenten innerhalb des Servers durchgereicht. Ist der E-Mail Server kompromittiert, so ist auch die Nachricht kompromittiert. Ist die

Nachricht allerdings GPG-verschlüsselt, so ist dies nicht der Fall. Zumindest nicht für den Body.

TLS

TLS bezeichnet das Protokoll »Transport Layer Security«, welches im RFC 5246 [11] definiert ist. Es ist ein Protokoll auf Applikationsebene, welches den Datenschutz und die Datenintegrität zweier kommunizierender Applikationen sicherstellen soll. Dabei besteht es aus zwei Schichten, einerseits aus dem »TLS Record Protocol« und dem »TLS Handshake Protocol«. (vgl. [11, S. 4])

Das »TLS Record Protocol« stellt dabei die unterste Schicht dar, direkt über dem TCP Protokoll und stellt sicher, dass die Verbindung selbst verschlüsselt ist. Dies findet über ein symmetrisches Verschlüsselungsverfahren statt. Dabei ist der gemeinsame Schlüssel auf die Laufzeit einer Sitzung begrenzt. Ferner definiert dieses Protokoll wie die Integrität von Nachrichten über Hashfunktionen wie SHA-1 sichergestellt wird. (vgl. [11, S. 4])

Das »TLS Handshake Protocol« hingegen ist im »TLS Record Protocol« eingebettet und übernimmt die Funktion der Authentifizierung. Auf dieser Ebene fällt ebenso die Entscheidung, welcher Verschlüsselungsalgorithmus benutzt wird und der Austausch der kryptografischen Schlüssel. Erst nach diesem Prozess wird die eigentliche Sitzung aktiv und Sitzungsdaten können gesendet werden. Dieser Prozess findet über asymmetrische Verschlüsselung basierend auf dem Public-Key Verfahren statt. Darauf basierend ist dann der Austausch des symmetrischen Sitzungsschlüssels sicher. (vgl. [11, S. 4])

Der öffentliche Schlüssel des Servers wird in der Praxis häufig von einem weiteren öffentlichen Schlüssel einer sogenannten »Certificate Authority« signiert. Die verbindende Gegenstelle muss also nur dem Schlüssel der Certificate Authority vertrauen. Applikationen wie Webbrowser liefern häufig ein Paket von vertrauten Schlüsseln solcher Certificate Authorities mit. Öffentliche Schlüssel können allerdings auch »self-signed« sein ohne Verbindung zu einer Certificate Authority. Diese müssen dann allerdings beim Empfänger manuell verifiziert und zugelassen werden. (vgl. [15, Kapitel 19 bis 21])

Die Schwächen und Probleme des TLS Protokolls und des darüber liegenden Zertifikats-Systems im einzelnen zu analysieren würde den Rahmen dieser Arbeit sprengen. Einige der bekannten Angriffe auf das Protokoll wurden in RFC 7457 [46] veröffentlicht. Für einige davon existieren Lösungen, entweder durch Konfigurationsänderungen am Server-System oder durch Erweiterungen der TLS Spezifikation. Festzuhalten ist aber lediglich, dass TLS das wohl am weitesten verbreitete Verfahren ist, um die Kommunikation im WWW, aber auch die Kommunikation von und zu E-Mail Servern zu

verschlüsseln und das Problem eines »Man-in-the-Middle-Angriff«, kurz MITMA [42], zu lösen.

Die Protokolle SMTPS, POP3S und IMAPS sind also nur Bezeichner für die Protokolle SMTP, POP3 und IMAP, die auf den dedizierten Ports 465, 995 und 993 ansprechbar sind und nur über das TLS Protokoll ansprechbar sind. Für diese existiert keine konkrete Spezifikation.

STARTTLS

STARTTLS ist eine Erweiterung für eine Reihe von Text-basierten Protokollen wie IMAP, POP3 und SMTP, aber auch FTP oder LDAP. Für IMAP und POP3 wurde diese in RFC 2595 [34] definiert und für SMTP in RFC 3207 [24]. Die Erweiterung benutzt das TLS Protokoll, hat aber im Vergleich zu einer gewöhnlichen TLS Kommunikation ein paar entscheidende Unterschiede:

- die Verbindung wird zunächst unverschlüsselt initiiert
- nach der Begrüßung antwortet der Server, dass er über STARTTLS verfügt und der Client kann eine TLS verschlüsselte Verbindung initiieren und anschließend z.B. Logindaten senden
- es wird kein dedizierter SSL Port benötigt
- STARTTLS ist allerdings protokollspezifisch

Nachfolgend eine SMTP-basierte STARTTLS Verbindung:

```
1  S: <wartet auf TCP Port 25>
2  C: <öffnet Verbindung>
3  S: 220 service ready
4  C: EHLO bauer.de
5  S: 250-bauer.de welcome
6  S: 250-STARTTLS
7  C: STARTTLS
8  S: 220 Go ahead
9  C: <startet TLS Verhandlung>
10 C & S: <Verhandlung einer TLS Sitzung>
11 C & S: <Überprüfen des Ergebnisses>
12 ...
```

Code 2.1: SMTP-STARTTLS

Ab Zeile 12 ist die Verbindung verschlüsselt und der Server kann beispielsweise über `AUTH PLAIN` eine Klartext-basierte Authentifizierung anfordern, da diese nun innerhalb der verschlüsselten Verbindung sicher ist.

Die Vorteile von STARTTLS sind einmal, dass keine zusätzlichen Ports benötigt werden, aber auch, dass optional eine unverschlüsselte Verbindung weitergeführt werden kann. Da dies aber häufig unerwünscht ist, ist es bedingt möglich den Server so zu konfigurieren, dass eine nachfolgende TLS Verbindung erforderlich ist (vgl. [34, S. 3]) (vgl. [24, S. 3]). Für POP3/IMAP in Verbindung mit STARTTLS müssen Server so konfigurierbar sein, dass eine erfolgreiche TLS Verbindung etabliert sein muss, bevor jede Art von Benutzerauthentifizierung stattfindet (vgl. [34, S. 3]). Auf SMTP Ebene ist es nur für Server erlaubt, die nicht »*publicly-referenced*« [24, S. 3] sind, TLS Verschlüsselung zu erzwingen.

Zu den Nachteilen gehört allerdings, dass bei STARTTLS mehr Metadaten verbreitet werden als bei direkter TLS Verbindung auf einem alternativen Port. Da die Verbindung initial unverschlüsselt ist, erlaubt dies auch weitere Angriffsvektoren wie z.b. »*STARTTLS Command Injection Attack (CVE-2011-0411)*« [46, S. 4], bei denen u.a. versucht werden kann, die Verbindung während der kurzen Periode von Klartextkommunikation herunterzustufen, indem das `STARTTLS` Kommando vom Client gar nicht erst den Server erreicht. Dies hängt aber auch wie schon eingangs erwähnt von der Serverkonfiguration ab. Ebenso sagt die Spezifikation nicht aus wie ein Client sich exakt zu verhalten hat, wenn eine STARTTLS Verbindung fehlschlägt: »*If the TLS negotiation fails or if the client receives a 454 response, the client has to decide what to do next. There are three main choices: go ahead with the rest of the SMTP session, retry TLS at a later time, or give up and return the mail to the sender.*« [24, S. 6]. Dies bedeutet, dass der Client theoretisch versuchen kann, mit einer Klartext-Verbindung fortzufahren. Dies muss auf Clientebene gelöst werden und entsprechende Konfigurationsoptionen angeboten werden, ist aber nicht garantiert.

Trotz der genannten Probleme ist STARTTLS das einzige Protokoll, das einen TLS-basierten Verbindungsaufbau von SMTP, POP3 und IMAP Servern überhaupt spezifiziert, im Gegensatz zu den Pseudoprotokollen SMTPS, POP3S und IMAPS, die keine konkrete Spezifikation haben und das Verhalten zwischen Client und Server demnach von Implementierungsdetails abhängt.

2.1.2 Anonymität von E-Mail-Verkehr

Wie bereits erwähnt bezeichnet Anonymität im Kontext dieser Arbeit das Verschleiern von Metadaten. Nicht immer ist die Trennung von Inhaltsdaten und Metadaten streng nachvollziehbar. So ist es beispielsweise diskutierbar, ob die allgemeinen Informationen einer SMTP Sitzung ohne Betrachtung der E-Mail Nachricht selbst als Metadaten gelten. Als Metadaten im Kontext dieses Kapitels sind aber vor allem Verbindungsdaten wie IP-Adressen gemeint.

Das SMTP Protokoll sowie die Protokolle POP3 und IMAP liefern über ihre Spezifikation keinerlei Methoden eine solche Anonymisierung durchzuführen. Ebenso sind keine Protokoll-Erweiterungen bekannt, die dieses leisten.

Allerdings sind protokollunabhängige Lösungen möglich. Eine davon ist die MUA-Erweiterung »TorBirdy«.

TorBirdy

»TorBirdy« [51] [49] ist eine Erweiterung für den MUA »*Mozilla Thunderbird*« [31] mit dem Ziel die Kommunikation folgender Komponenten zu anonymisieren:

- MUA kommuniziert mit entfernten MSA
- MUA kommuniziert mit entfernten POP3/IMAP4 Server

Dies sind zwei von den eingangs drei erwähnten Angriffspunkten.

Das Prinzip von TorBirdy ist, den gesamten Datenverkehr der Kommunikation von Thunderbird mit dem E-Mail Server über das Tor-Netzwerk [12] zu leiten.

Das Tor-Netzwerk selbst ist allerdings abstrakter als dieser Anwendungsfall. Es erlaubt allgemein das Anonymisieren von TCP-Verbindungen jeglicher Art. Dieses wird wiederum auf Basis des »Onion Routings« [22] [13] realisiert. Das Onion Routing läuft grob in folgenden Schritten ab (vgl. [13]):

1. der initiale Sender wählt von einem sogenannten Verzeichnisserver (engl. »directory node«) eine geordnete, aber zufällige Liste von Knoten aus, welche den Pfad bilden, den die Daten bis zum schlussendlichen Empfänger nehmen
2. vom Verzeichnisserver wird ein öffentlicher Schlüssel (asymmetrisches Verschlüsselungsverfahren) für den ersten Knoten bezogen, der sogenannte Eintrittsknoten (engl. »entry node«)
3. eine Verbindung wird hergestellt und ein Sitzungsschlüssel (symmetrisches Verschlüsselungsverfahren) erstellt
4. über die erstellte Verbindung kann der Sender eine Nachricht an einen zweiten Knoten senden, die allerdings mit dessen öffentlichem Schlüssel verschlüsselt wurde und somit nicht vom Eintrittsknoten entschlüsselt werden kann
5. der zweite Knoten verbindet sich dazu mit dem ersten Knoten und ist somit indirekt mit dem Sender verbunden, ohne davon Wissen zu haben
6. diese Verbindung kann jetzt zu einem dritten, vierten, usw. ... Knoten erweitert werden

7. der schlussendliche Empfänger benutzt dieselbe Reihe von Knoten zurück zum Sender, nur umgekehrt

Dieses Konzept wird »Onion Routing« genannt, da hier eine verschachtelte Verschlüsselung vorliegt, die nach und nach beim Weiterreichen an den jeweils nächsten Knoten wie eine Zwiebel geschält wird. Die Nachrichten für jede Schicht sind mit dem öffentlichen Schlüssel für exakt diese Schicht verschlüsselt, sodass kein Knoten in der Mitte den vollständigen Pfad herausfinden kann. Zu jedem Zeitpunkt hat jeder Knoten nur Wissen über den vorherigen Knoten und den nachfolgenden Knoten. Der erste Knoten ist immer der Eintrittsknoten und der letzte der Austrittsknoten (engl. »exit node«). Die Verbindungspfade werden im Tor-Netzwerk in regelmäßigen Abständen geändert. Das Tor-Netzwerk selber benutzt intern konsistent TLS Verschlüsselung.

Dieses Konzept wird vor allem im WWW benutzt, über den sogenannten »Tor Browser« [50]. TorBirdy macht allerdings in derselben Weise Gebrauch davon, bleibt aber kompatibel mit den Protokollen SMTP, POP3 und IMAP.

Anzumerken ist, dass der Versand von E-Mails auch über Weboberflächen möglich ist und somit eine gewöhnliche Verbindung über den Tor Browser verwendet werden kann. Allerdings gibt es auch diverse Gründe dies für vollwertige MUAs zu ermöglichen (vgl. [49, S. 4]).

2.1.3 Bewertung

Nahezu alle Datenschutzmechanismen für E-Mails sind erst wesentlich später als die ursprünglichen Protokolle wie SMTP entstanden und haben sich mit der Zeit entwickelt. Viele der vorgestellten Technologien sind unabhängige Lösungen, die lediglich in Verbindung mit E-Mail Systemen verwendet werden können, andere sind wiederum Protokollerweiterungen, die sich mehr oder weniger sinnvoll in das Gesamtkonzept einfügen. Sichtbar anhand der untersuchten Technologien ist allerdings, dass hier keine konsistente Lösung vorliegt. Jede Lösung zielt auf einen anderen Bereich ab und versucht ein atomares Problem zu lösen, ohne dabei auf die Kohärenz eines in der Praxis eingesetzten E-Mail Systems achten zu können. Dies öffnet unabhängig vom Kohärenzproblem auch das Problem der Konfiguration eines konkreten Systems. Die immense Komplexität der zusammenspielenden Komponenten wurde in Kapitel 1 erhellt und ist ein praktisches Problem solcher Systeme.

Dennoch ist es möglich die genannten Systeme zu nutzen, um eine Reihe von Problemen bezüglich Datensicherheit zu lösen. Um die Inhaltsdaten des Bodies einer E-Mail für jeden außer der Empfängerperson unzugänglich zu machen, kann das asymmetrische Verschlüsselungsverfahren GPG benutzt werden. Selbst der Server-Betreiber oder ein Man-in-the-Middle ist dann nicht in der Lage, die E-Mail ohne weiteres zu entschlüsseln, unabhängig davon, in welcher Weise diese transportiert wurde. Allerdings

bezieht sich GPG nur auf den Body einer E-Mail und nicht auf die Header. Ebenso ist es möglich, die Protokoll-Transaktionen von SMTP, POP3 und IMAP über direktes TLS oder STARTTLS zu verschlüsseln. Problematisch hierbei ist aber, dass eine verschlüsselte Verbindung nicht durchgängig garantiert werden kann, z.b. wenn die Verbindung über mehrere MTAs geht oder wenn der E-Mail Server erlaubt zu unverschlüsselten Verbindungen herunterzustufen, insbesondere über SMTP STARTTLS. Und letzten Endes ist eine gewisse Anonymisierung der IP-Verbindungsdaten über TorBirdy möglich. Diese gilt allerdings nur für die Kommunikation des MUA mit dem MSA bzw. POP3/IMAP server.

Abbildung 2.2 zeigt sehr deutlich, dass selbst beim Einsatz all dieser Technologien und unter der Voraussetzung korrekter Konfiguration immer noch eine wesentliche Stelle nicht ausreichend anonymisiert oder verschlüsselt werden kann. Dies ist nämlich die Kommunikation zwischen mehreren MTAs. Hier kann durchgängig nicht angenommen werden, dass Transaktionen verschlüsselt sind. Ebenso sind die Verbindungen direkt und weiterleitende E-Mail Server werden über den `Received` Header gekennzeichnet. Somit ist es ohne viel Aufwand möglich, den Weg, den eine E-Mail nimmt, nachzuverfolgen und zumindest die kommunizierenden E-Mail Server eindeutig zu bestimmen. Die Menge an offenen Metadaten an dieser Stelle ist so hoch, dass hier keine Gesamtlösung zu erkennen ist und u.U. Sender und Empfänger direkt identifiziert werden können.

Allerdings ist festzuhalten, dass abgesehen von der mangelnden Verschleierung der Kommunikation zwischen MTAs, die existierenden Technologien die Basis einer Gesamtlösung bilden können.

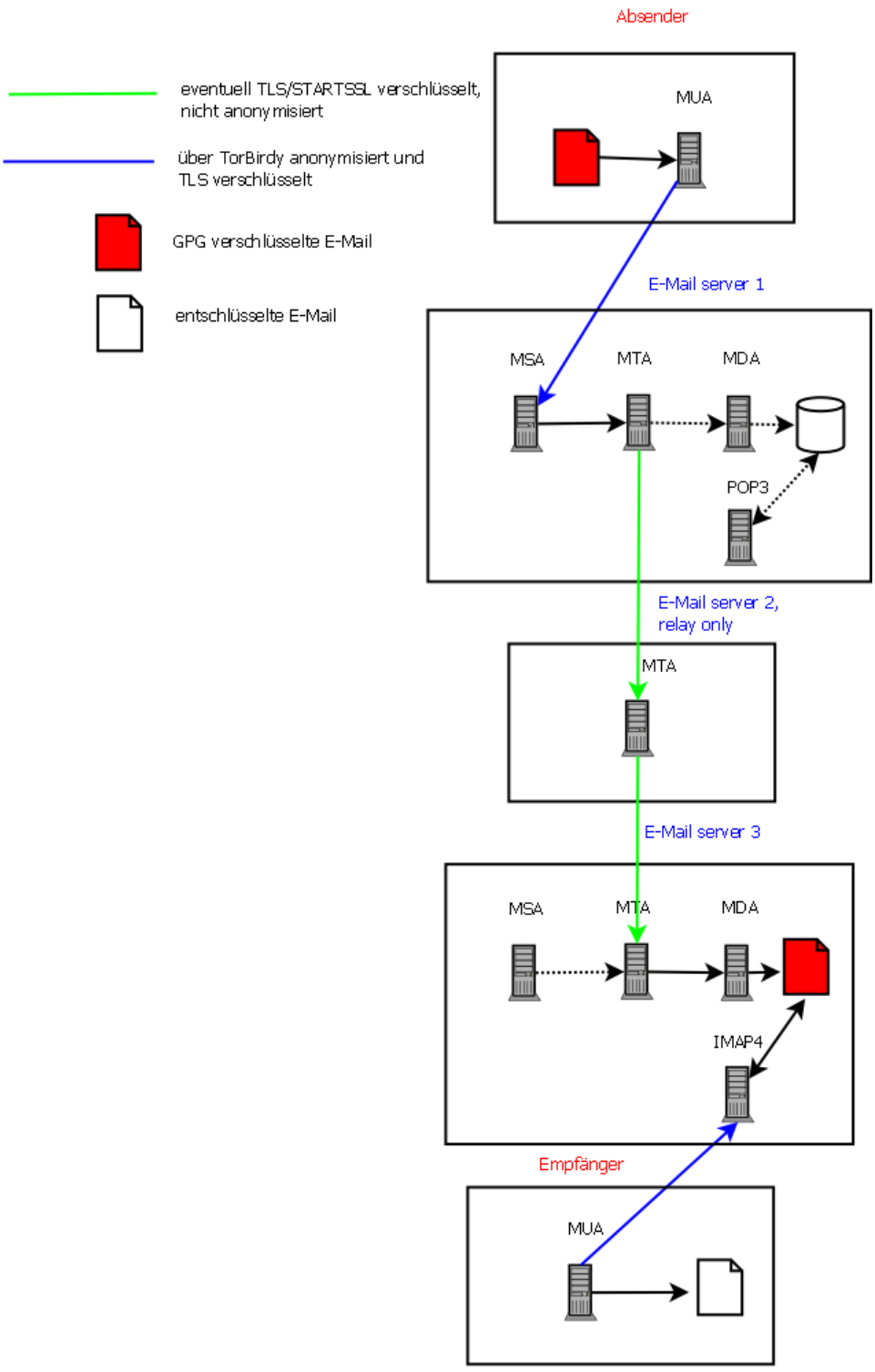


Abbildung 2.2: Übersicht Datenschutz in E-Mail Systemen

2.2 Anforderungen an eine adäquate Lösung

Nachfolgend werden die Anforderungen an eine adäquate Lösung formuliert. Dabei handelt es sich nicht um die Anforderungen an ein E-Mail System an sich, sondern um die Anforderungen, die relevant für größtmögliche Datensicherheit und Anonymität in einem bereits funktionierenden E-Mail System sind. Als System werden alle Komponenten bezeichnet, die beim Senden und Empfangen einer E-Mail involviert sind.

Die nachfolgenden Anforderungen sind rein funktional und beziehen sich lediglich auf Verschlüsselung und Anonymisierung.

Nummer	Anforderung
FA 1	Das System muss dem Benutzer die Möglichkeit bieten, den Textinhalt seiner E-Mail vor dem Versenden an den MSA kryptografisch zu verschlüsseln.
FA 2	Das System muss dem Benutzer die Möglichkeit bieten, die Transaktion zwischen MUA und MSA über eine verschlüsselte Verbindung zu etablieren.
FA 3	Das System muss dem Benutzer die Möglichkeit bieten, die Verbindungsdaten der Transaktion zwischen MUA und MSA zu anonymisieren.
FA 4	Das System muss dem Benutzer die Möglichkeit bieten, die Transaktion zwischen MUA und POP3/IMAP Server über eine verschlüsselte Verbindung zu etablieren.
FA 5	Das System muss dem Benutzer die Möglichkeit bieten, die Verbindungsdaten der Transaktion zwischen MUA und POP3/IMAP Server zu anonymisieren.
FA 6	Das System muss dem Benutzer die Möglichkeit bieten, den Versand und die Zustellung der E-Mail auf MTA-Ebene nur über verschlüsselte Verbindungen zu erlauben.
FA 7	Das System muss dem Benutzer die Möglichkeit bieten, den Versand und die Zustellung der E-Mail auf MTA-Ebene nur über anonymisierte Verbindungen zu erlauben.

Tabelle 2.1: Funktionale Anforderungen

Die nachfolgenden Anforderungen sind nicht funktional und beziehen sich auf mögliche Probleme, die aus der Implementierung der genannten funktionalen Anforderungen entstehen könnten.

Nummer	Anforderung
NFA 1	Das System muss kompatibel mit den Protokollen SMTP, POP3 und IMAP sein.
NFA 2	Das System sollte die bereits existierenden Methoden zur Spam-Abwehr nicht erschweren.

Tabelle 2.2: Nicht-Funktionale Anforderungen

Wie zu erkennen ist, folgt aus den funktionalen Anforderungen der Versuch, nicht nur Teile des E-Mail Verkehrs zu verschlüsseln und anonymisieren, sondern das System als ganzes. Da das betrachtete System bestehend aus den Protokollen SMTP, POP3 und IMAP aber aus verschiedenen Einzelteilen besteht, müssen die Lösungen auch spezifisch auf die jeweiligen Teilsysteme bezogen sein. Das bedeutet ebenso, dass ein direkter Bezug zu konkreten Protokollen besteht und dies keine abstrakte Anforderungsliste an irgendein E-Mail System ist.

Die nichtfunktionalen Anforderungen beschäftigen sich hingegen mit der Kompatibilität und der praktischen Einsetzbarkeit des Systems. Diese Komponenten spielen eine wichtige Rolle für eine mögliche Akzeptanz einer Lösung.

Kapitel 3

Alternative Lösungen bzw. E-Mail Systeme

Zur Lösung der Datenschutzprobleme auf SMTP-Ebene existieren wenig wissenschaftliche Arbeiten. Eine etwas theoretische Arbeit ist »Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms« [7], welche allerdings nicht auf konkrete E-Mail Protokolle eingeht, da sie auch ca. 1 Jahr vor dem ersten SMTP Standard [39] veröffentlicht wurde.

Allerdings existieren alternative E-Mail Systeme, die das Problem des Datenschutzes auf ihre eigene Weise lösen ohne an SMTP gebunden zu sein. Dazu gehören beispielsweise Dark Mail [7], aber auch Bitmessage [54], welches auf der Bitcoin [33] Technologie basiert.

Nachfolgend wird Dark Mail kurz besprochen.

3.1 Dark Mail

Als sehr genau spezifizierte Alternative zur E-Mail muss Dark Mail, bzw. »Dark Internet Mail Environment« [27] genannt werden. Dark Mail verwirft jegliche bekannten Protokolle wie SMTP und entwickelt ein gänzlich neues E-Mail Konzept mit den Protokollen »*Dark Mail Transfer Protocol*« [27, S. 82-87] und »*Dark Mail Access Protocol*« [27, S. 108]. Ebenso kommt ein eigenes Datenformat namens »*D/MIME*« [27, S. 68-81] zum Einsatz.

Auf die genauen Einzelheiten kann hier nicht eingegangen werden, da Dark Mail sehr komplex ist. Wichtig zu bemerken ist allerdings, dass es eine dem Onion Routing [22] [13] ähnliche Methodik benutzt. Ebenso kommt verschachtelte Verschlüsselung und ein durchdachtes Schlüsselsystem zum Einsatz. Allerdings ist Dark Mail auf Implementierungsebene noch nicht fertig und es existiert zum derzeitigen Zeitpunkt kein Release [2].

3.2 Bewertung

Alternative Lösungen wie Dark Mail [7] oder Bitmessage [54] sind zum Teil sehr elegant und durchdacht, entfernen sich aber deutlich von den weitverbreiteten Protokollen wie SMTP oder IMAP. Dies mag zwar notwendig sein, um diverse Spezifikationsfehler in den genannten Protokollen zu überwinden, ist aber gleichzeitig auch ein Problem. Denn die neuen vorgeschlagenen Protokolle müssen mit den bestehenden Protokollen konkurrieren, da sie mit diesen inkompatibel sind. Ob diese Ideen genügend Penetrationspotenzial haben, um von der Internetgemeinde angenommen zu werden, bleibt fraglich.

Kapitel 4

Entwicklung eines SMTP-basierten Anonymisierungs-Algorithmus

4.1 Idee & Konzept

Zur Lösung der Anforderungen in Abschnitt 2.1 wird nachfolgend ein Algorithmus diskutiert und vorgestellt, der die Verbindung zwischen MTAs auf SMTP-Ebene anonymisieren soll, sodass Sender und Empfänger einer E-Mail nicht durch Verfolgung von IP-Adressdaten nachvollzogen werden können.

Bei der Überlegung eines solchen Algorithmus spielt vor allem Abschnitt NFA 1 eine Rolle. Alternative E-Mail Systeme mit einem Schwerpunkt auf Datensicherheit wurden bereits in Kapitel 3 vorgestellt. Im Zuge dieser Arbeit wird allerdings versucht, der Akzeptanz halber eine SMTP-basierte Lösung zu entwickeln. Dies soll, mit minimalem Aufwand für E-Mail Server Betreiber und Nutzer, eine Verbesserung der Datensicherheit ermöglichen.

Bei der Wahl und Entwicklung des Algorithmus wurde vor allem Wert darauf gelegt, dass keine spezifischen SMTP Erweiterungen notwendig sind. Somit soll dieser auf dem Minimum des IMF und SMTP Protokolls fußen.

Die Anonymisierung soll über zufälliges Routing realisiert werden, welches kryptografisch gestützt wird. Alle Informationen zum Routing sollen in der E-Mail kodiert sein. Jeder Knoten soll zu jedem Zeitpunkt nur den Nachfolger und Vorgänger kennen. Die originale E-Mail darf nicht modifiziert werden, was bedeutet, dass ein Verschachtelungsverfahren angewendet werden muss.

4.1.1 Algorithmus

Der Algorithmus wird nachfolgend »MystMail Algorithmus« genannt und ist vom Grundprinzip dem Onion Routing [13] nicht unähnlich.

Der Algorithmus bezieht sich allerdings nicht nur auf MTAs, sondern muss auch auf der Ebene der Kommunikation zwischen MUA und MSA eingreifen.

In ihrer Funktionsweise ist die MystMail eine gewöhnliche E-Mail, die eine Art Liste von Relay Servern beinhaltet. Die E-Mail wird also nicht direkt an den schlussendlichen Empfänger gesendet, sondern über mehrere Relay Server (auch »Hops« genannt) geleitet, basierend auf den kodierten Informationen in der E-Mail. Dies ist der Anonymisierungspfad. Damit dieser Pfad selbst weder für die Hops noch für irgendjemand sonst sichtbar ist, muss er kryptografisch in die E-Mail eingebettet werden.

Die sogenannte Liste von Hops ist also keine Liste, die in Form von Klartext als Header festgesetzt wird. Stattdessen wird die originale E-Mail inklusive Header über ein asymmetrisches Verschlüsselungsverfahren mit dem öffentlichen Schlüssel des ersten Hops verschlüsselt und fungiert jetzt als Body einer neuen E-Mail, die an denselben Hop gerichtet ist, gewissermaßen die erste MystMail. Für den nächsten Hop wird das Verfahren mit der soeben neu erstellten MystMail wiederholt. Dies kann theoretisch beliebig oft wiederholt werden und hat zur Folge, dass eine verschachtelte Verschlüsselung vorliegt, jede Schicht mit einem anderen öffentlichen Schlüssel verschlüsselt. Dies bedeutet wiederum, dass kein Hop in der Mitte des Pfades an die übernächste MystMail gelangen kann, da er nicht den privaten Schlüssel dafür besitzt. Somit kennt jeder Hop nur Vor -und Nachfolgehop.

Ist nun die schlussendliche MystMail mit der impliziten Liste von Hops vollständig durch den MUA erstellt, wird sie an den ersten Hop gesendet. Dieser entschlüsselt den Body mit seinem privaten Schlüssel und erhält die nächste MystMail mit Informationen zum nächsten Hop im `To:` Headerfeld. Diese MystMail versendet er über SMTP nun an den nächsten Hop, welcher die nächste MystMail entschlüsselt, usw. Die MystMail hat jedoch ein dediziertes Headerfeld `X-Myst-Mail: 1`, der SMTP-kompatibel markiert, dass diese E-Mail eine spezielle E-Mail ist. Durch den ganzen Pfad ist dieses Headerfeld aktiv. Gelangt die E-Mail an den schlussendlichen Empfänger, so entschlüsselt dieser den Body der E-Mail ohne bereits zu wissen, dass er der Empfänger ist. Erst nachdem der MTA keinen `X-Myst-Mail: 1` findet, weiß er, dass er der schlussendliche Empfänger ist und kann die entschlüsselte E-Mail lokal über den MDA zustellen.

Dieser Algorithmus wird auf MTA-Ebene durch das `X-Myst-Mail: 1` Headerfeld ausgelöst. Ist dieses nicht Bestandteil des Headers, so wird die E-Mail nach den gewöhnlichen Regeln des MTAs zugestellt oder weitergeleitet. Die Art und Weise, wie dieser Algorithmus auf MUA Ebene ausgelöst wird, ist nicht spezifiziert.

Sowohl der Verschlüsselungsalgorithmus für die verschachtelten E-Mails als auch der Zufallsalgorithmus zur Auswahl der Hops wird hier nicht spezifiziert.

In chronologischen Schritten kann der Algorithmus folgendermaßen dargestellt werden:

```
1  MUA (Sender): empfängt Liste von Hops von einem zentralen
2                Server
3  MUA (Sender): wählt aus der Liste einen zufälligen Pfad aus
4  MUA (Sender): erstellt die initiale verschachtelte MystMail
5                für den ersten Hop
6  MUA (Sender): reicht die MystMail an seinen MSA ein
7  MSA (Sender): nimmt MystMail entgegen
8  MTA (Sender): sendet die MystMail an den ersten Hop
9  MTA (1. hop): nimmt die MystMail entgegen
10 MTA (1. hop): erkennt den "X-Myst-Mail: 1" Header
11 MTA (1. hop): entschlüsselt den Body der E-Mail mit seinem
12               öffentlichen Schlüssel
13 MTA (1. hop): erkennt, dass auch dort ein "X-Myst-Mail: 1"
14               Header vorliegt
15 MTA (1. hop): benutzt den entschlüsselten Text als neue E-
      Mail
16               und sendet die MystMail an die Adresse aus
      dem
17               "To:" Headerfeld, den 2. Hop
18 MTA (2. hop): nimmt die MystMail entgegen
19 [...] (wie 1. hop)
20 MTA (3. hop): nimmt die MystMail entgegen
21 [...] (wie 1. hop)
22 MTA (Empfänger): nimmt die MystMail entgegen
23 MTA (Empfänger): erkennt das "X-Myst-Mail: 1" Headerfeld
24 MTA (Empfänger): entschlüsselt den Body der E-Mail mit
      seinem
25                  öffentlichen Schlüssel
26 MTA (Empfänger): erkennt, dass im entschlüsselten Text kein
27                  "X-Myst-Mail: 1" Headerfeld vorliegt
28 MTA (Empfänger): leitet die E-Mail an den MDA weiter
29 MDA (Empfänger): stellt lokal zu
30 MUA (Empfänger): ruft E-Mail vom POP3/IMAP Server ab
31                  und sieht die original E-Mail
32                  mit dem korrekten "From:" Feld vom Empfä
      nger
```

Code 4.1: MystMail-Algorithmus Simulation

Auf Basis des MTA ist also eine Entscheidung notwendig, abhängig vom `X-Myst-Mail` Headerfeld. Diese Entscheidung kann folgendermaßen dargestellt werden:

```
1  // either relays a MystMail or delivers the inputmail in
       case
2  // it is not a MystMail
3  function processMail (inputmail) {
4    header = inputmail.header;
5    body = inputmail.body;
6    if header.getField("X-Myst-Mail").notNull()
7      newMystMail = decrypt(body);
8      envelope = createEnvelope(newMystMail);
9      send(newEnvelope);
10   else
11     deliverToMDA(inputmail);
12 }
13
14 // creates the Envelope that carries Meta-Data for the
15 // SMTP transaction, such as "RCPT TO" and "MAIL FROM"
16 function createEnvelope(inputmail) -> Envelope {
17   from = inputmail.header.getField("From");
18   to = inputmail.header.getField("To");
19
20   return new Envelope(mail = inputmail
21                       , RCTP = [to]
22                       , MAILFROM = from);
23 }
```

Code 4.2: MystMail MTA Entscheider

Für den MUA liegt folgendes Verhalten vor:

```
1  // creates the initial MystMail
2  function createMystMail(inputmail) -> EMail {
3    hops = getHopList();
4    path = chooseHopPath(hops) ++ inputmail.recipient;
5    newmail = inputmail;
6
7    for hop in path {Alle Informationen zum routing sollen in
       der E-Mail kodiert sein.
8      if (hop == path[0])
9        from = "myst@" ++ inputmail.sender.address.domain;
10     else
11       from = hop.prev.address;
12     to = "myst@" ++ hop.domain;
13     body = encrypt(hop, newmail);
14     newmail = EMail(From = from
15                     , To = to
16                     ).addHeader("X-Myst-Mail: 1");
17   }
18
19   return newmail;
20 }
```

Code 4.3: MystMail-Erstellung (MUA)

Abbildung 4.1 zeigt schematisch den Ablauf des Algorithmus. Die initiale E-Mail ist 4-fach verschachtelt verschlüsselt und geht über 3 Hops. Jeder Hop entschlüsselt den Body der eingegangenen E-Mail, benutzt diesen entschlüsselten Text als vollständige IMF-kompatible E-Mail inklusive Header und Body und leitet diese dann an den nächsten Hop weiter. Der Empfänger MTA erkennt, dass kein `X-Myst-Mail` Headerfeld vorliegt und reicht die unveränderte originale E-Mail an den MDA weiter. Die farbig markierten Pfeile zeigen an, welcher Teil der MystMail gerade weitergesendet wird.

Anzumerken ist hierbei, dass abgesehen von der originalen E-Mail alle Header das `From` und `To` Feld verschleiern, indem der Benutzername `myst` verwendet wird. Technisch gesehen ist es unerheblich, wie der Benutzername definiert ist, da dies das Verhalten des MystMail MTAs nicht verändert.

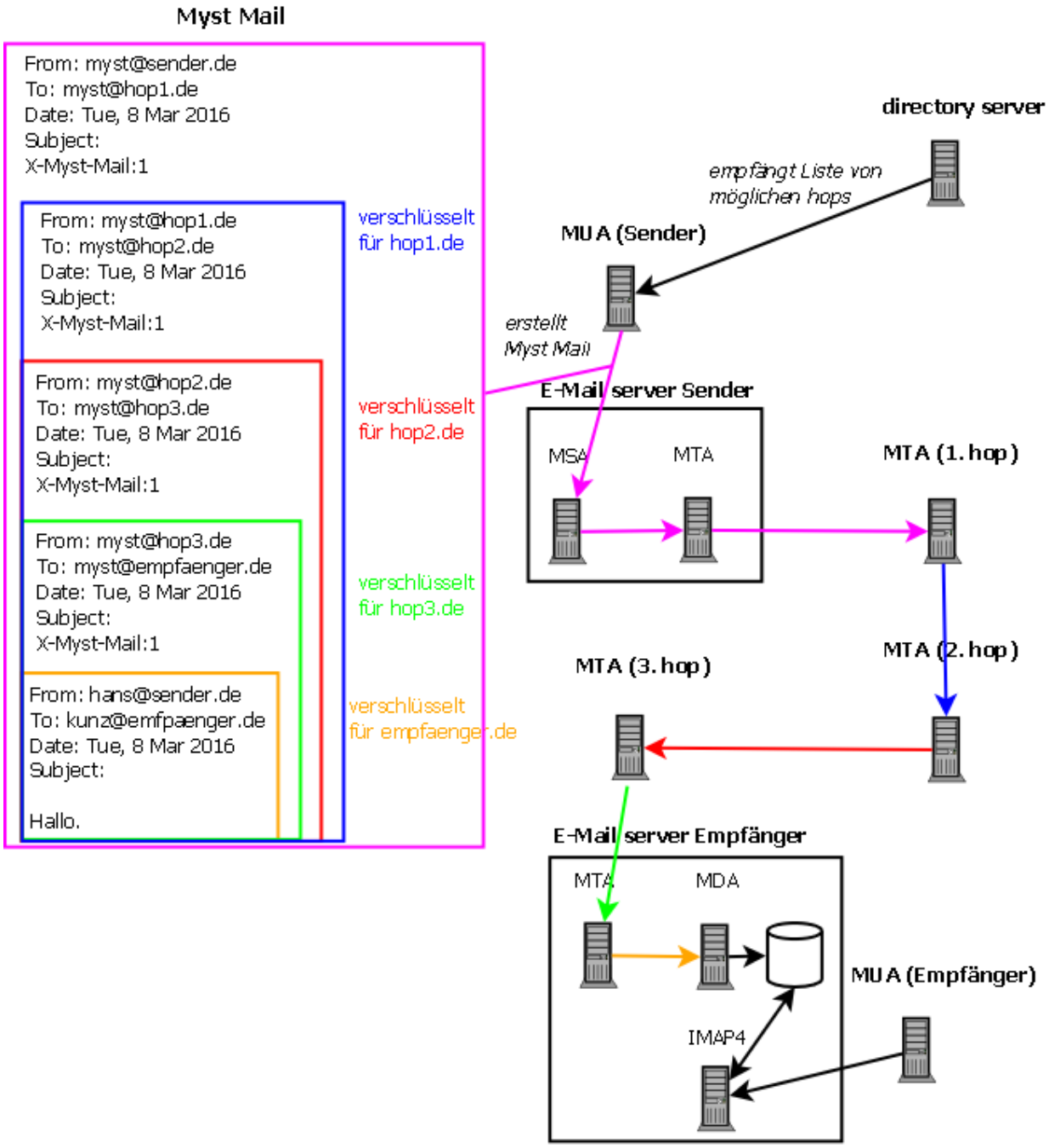


Abbildung 4.1: Übersicht MystMail Algorithmus

4.2 Methoden

Die hier beschriebenen Methoden beziehen sich auf die wissenschaftliche Arbeitsweise, sowohl auf theoretischer als auch auf praktischer Ebene, welche in dieser Arbeit Anwendung findet.

Dem Geiste nach wird versucht dem wissenschaftstheoretischen Ideal aus der »Logik der Forschung« [38] von Karl R. Popper zu entsprechen. Demnach ist die Arbeit als Ganzes vor allem ein Diskussionsvorschlag an den wissenschaftlichen Apparat. Sie stellt aber ebenso eine Behauptung auf mit der Erwartung, dass an derselben Falsifikationsversuche durchgeführt werden können.

Das Wesen der Falsifikation ist hier allerdings etwas anders als in klassischen empirischen Wissenschaften wie der Physik. Neben der logischen Falsifikation, die sich vor allem auf Argumentationsfehler und Logikfehler im Allgemeinen bezieht, ist die sogenannte empirische Falsifikation im Kontext der hier aufgestellten Behauptungen weniger exakt und die Trennung beider Formen weniger eindeutig. Konzeptionelle Fehler und Schwächen können sowohl a priori, als auch a posteriori erkannt werden, da die Tauglichkeit des vorgeschlagenen Systems nicht rein formal bestimmt werden kann. Viele in der Praxis relevante Details und Faktoren wirken auf das System als ganzes ein und machen Falsifikationsversuche schwieriger. Mögliche Angriffsvektoren werden häufig erst entwickelt oder entdeckt, wenn ein System bereits eine Weile in Benutzung ist. Gerade deshalb sind weite Teile des vorgeschlagenen Systems nur sehr abstrakt beschrieben. Somit konzentriert sich diese Arbeit mehr auf eine Idee, weniger auf ein vollständig durchdachtes Konzept und kann letzteres auch nicht leisten.

Die genannte Idee wird im Zuge dessen argumentativ unterstützt, sowohl im Rückblick auf das existierende E-Mail System und seine Schwächen, als auch im Hinblick auf Systeme im allgemeinen, die ähnliche algorithmische Methoden erfolgreich verwenden.

Auf dieser Basis wird weiterhin versucht nachzuweisen, dass der Algorithmus im Kern implementierbar ist und die Proof of Concept Implementierung zumindest die wichtigsten Anforderungen aus Abschnitt 2.1 und Abschnitt 2.2 tatsächlich erfüllen kann.

Die Implementierung wird sich auf den Kern-Algorithmus beschränken und ein minimales Testsystem zur Simulation von MTA-Kommunikation beinhalten.

Bei der Entwicklung der Implementierung wird eine abgewandelte Form des Prototyping Modells [28] angewendet. Dabei ist der erste Schritt die Entwicklung eines Testsystems, das ein verteiltes System simulieren kann. Andernfalls ist es schwierig, in kurzen Abständen Codeänderungen durchzuführen und deren Auswirkungen zu überprüfen. Sobald dieses Testsystem steht, wird eine existierende SMTP Implementierung ausgewählt und diese ins Testsystem integriert. Das Testsystem beschränkt sich hier-

bei auf sichtbare Laufzeittests. Ist diese grundlegende Funktionalität gegeben, so wird der eigentliche Algorithmus injiziert und ohne weitere Iterationen vervollständigt. Erst nach dieser Phase findet eine Reflexion statt, die Refactoring oder sogar eine vollständige Neuentwicklung zur Folge haben kann. Der Dokumentationsprozess ist ein fortlaufender Prozess, der in jeder Phase der Entwicklung stattfindet.

4.3 Realisierung

Die Realisierung auf Basis des Prototyping Modells durchläuft mehrere Phasen. Zunächst wird ein Testsystem entwickelt, das auf Docker [14] basiert. Dieses Testsystem wird es erlauben, mehrere MTAs miteinander isoliert kommunizieren zu lassen. Dafür muss ebenso eine SMTP Implementierung gewählt werden. Nach der Entwicklung des Testsystems wird der Einsprungpunkt gesucht, an dem der Algorithmus in das bestehende SMTP Protokoll injiziert werden kann. Ist dieser gefunden, wird zunächst der MystMail MTA Entscheider (Code 4.2) implementiert. Dieser wird noch nicht die Logik für das Erstellen der weiterzuleitenden MystMail beinhalten oder das Weiterleiten selbst, sondern lediglich die Entscheidungslogik testen. Danach kann überprüft werden, ob das SMTP Protokoll immer noch einwandfrei funktioniert und beim Erkennen einer MystMail der entsprechende Entscheidungszweig erreicht wird. Ist die Korrektheit hier verifiziert, wird der Teil des MystMail MTA Entscheiders implementiert, der die Nachricht, zunächst noch unverändert, an einen vordefinierten MTA weiterleitet. Somit wird das Weiterleiten zwischen den MTAs getestet. Ist das Weiterleiten möglich, so wird die eigentliche Logik zum Entpacken einer MystMail implementiert, inklusive dem Interpretieren des Bodies und dem Erstellen eines Envelope. Nach diesem Schritt wird eine statische initiale MystMail per Hand erstellt und das Entpacken sowie die Weiterleitung zwischen den MTAs getestet. Als letzter Schritt wird die MystMail-Erstellung (Code 4.3) als SMTP-Gateway implementiert, nicht als MUA Erweiterung. Dies geschieht aus Gründen der Einfachheit.

Nachfolgend werden die einzelnen Phasen der Entwicklung beschrieben, Entscheidungen erklärt und Probleme erläutert.

4.3.1 Testsystem

Für das Testsystem wird eine isolierte Umgebung benötigt, die ausreichend Kontrolle über Netzwerkdetails erlaubt und trotzdem möglichst frei von etwaigen Netzwerkproblemen ist. Würde hier echte Hardware eingesetzt werden, so käme das Problem hinzu, dass auf Hardware-Ebene Probleme und Fehler das System beeinträchtigen könnten. Eine manuelle lokale Konfiguration innerhalb eines Host-Systems, die das simulieren mehrerer MTAs erlaubt, ist aufgrund der Konfigurationskomplexität eben-

so unerwünscht. Ebenso würde dies eine Abhängigkeit vom Host-System und dessen Konfiguration bedeuten. Aus diesem Grund wird eine Container-Software eingesetzt, die die entsprechenden Abstraktionsschichten bereits bereitstellt. Sie erlaubt mehrere gleichartige Prozesse isoliert laufen zu lassen und dieselben über eine definierte softwareseitige Netzwerkschnittstelle kommunizieren zu lassen. Für diese Art von Software existieren mehrere Lösungen. Für die Realisierung wird hier allerdings die Software »Docker« [14] eingesetzt.

Docker ist keine Hardwarevirtualisierung. Es ist eine Abstraktionsschicht, die es erlaubt Prozesse in isolierten User-Spaces auf demselben Betriebssystemkernel laufen zu lassen. Dazu benutzt es Kernel Features wie Cgroups und Kernel Namespaces. Auf Dateisystem-Ebene wird UnionFS benutzt.

Die praktische Funktionsweise reduziert sich auf folgenden Ablauf: Der Benutzer erstellt eine Definitionsdatei mit dem Namen »Dockerfile«. Diese definiert gewissermaßen den Installationsprozess eines vollständigen Betriebssystems, der darin benötigten Applikationen, sowie die Art und Weise wie diese ausgeführt und konfiguriert werden. Auf Basis dieses Dockerfiles kann ein »Docker Image« erstellt werden. Ein Docker Image kann als Vorlage betrachtet werden, mit der eine Chroot-ähnliche Umgebung erstellt und ausgeführt werden kann. Diese ausgeführte Umgebung wird »Docker Container« genannt. Wichtig ist dabei, zu wissen, dass das Image aus mehreren UnionFS-Layern besteht (pro Dockerfile-Anweisung ein Layer). Wenn ein Container gestartet wird, so wird auf die Daten aus dem Image zugegriffen. Allerdings sind diese immer read-only. Sobald Änderungen an Dateien innerhalb eines Containers gemacht werden, wird die read-only Schicht des Images mit einer read-write Schicht des laufenden Containers zusammengeführt, ohne dass die Daten des Images sich dabei verändern. Somit können auf Basis eines beispielsweise gemeinsamen MySQL-Images mehrere MySQL-Instanzen gestartet werden, die alle dieselbe Version, dasselbe Betriebssystem und dieselbe Umgebung besitzen, aber dennoch voneinander isoliert sind, sowohl auf Prozess-, Netzwerk- und Dateisystem-Ebene. Beim Starten von Containern ist es allerdings möglich, diese miteinander zu verbinden, so dass sie auf Netzwerkebene miteinander kommunizieren können.

Mithilfe dieses Systems wird es möglich, mehrere gleiche MTAs zu starten und zu verbinden. Dabei können diese sich dennoch in der Konfiguration unterscheiden, da im laufenden Container Änderungen vorgenommen werden können. Ebenso ist es möglich, über Umgebungsvariablen oder Mountpoints, die vom Host in den Container zeigen, den Zustand und die Konfiguration eines Containers zu steuern. Ebenso sind die Container nahezu vollständig unabhängig vom laufenden Host-System, da ein Container bereits ein vollständiges Betriebssystem darstellt und lediglich mit dem Kernel des Host-Systems über den Docker Daemon kommuniziert wird.

Es werden gemäß Abbildung 4.1 also 5 E-Mail Container gestartet. Alle 3 Hops und der E-Mail Server des Empfängers haben dieselbe Konfiguration. Nur der E-Mail Server des Senders hat eine abweichende Konfiguration, da dieser als SMTP Gateway fungiert und anstelle des MUAs die initiale MystMail generiert. Alle Container werden in einem Netzwerk zusammengefasst und können untereinander kommunizieren.

Dieses System erlaubt es nun, fortlaufend Laufzeittests durchzuführen. Diese finden nicht automatisiert statt, sondern werden über die laufenden Prozesse in den Containern und deren Logdateien manuell durchgeführt.

Die Implementierung des Testsysems ist im Anhang einsehbar.

4.3.2 Implementierung

Für die Implementierung wurden zunächst verschiedene SMTP-Softwarelösungen betrachtet und eine adäquate ausgewählt. Nachfolgend werden systematisch, wie in Kapitel 4.3 erklärt, die einzelnen Implementierungsschritte durchlaufen.

Wahl der SMTP-Software

Bei der Wahl der SMTP-Software waren mehrere Gesichtspunkte von besonderer Bedeutung:

- wird die Software tatsächlich eingesetzt?
- wird die Software aktiv entwickelt und gepflegt?
- in welcher Sprache ist die Software geschrieben?
- ist die Software (ausreichend) dokumentiert?
- wie kompliziert ist die Konfiguration?
- wie einfach lässt sich die innere Logik verändern?

Diese Punkte sind wichtig, um eine mögliche Weiterentwicklung der Idee auf Basis derselben Implementierung einfacher zu machen, obwohl diese nur eine Proof-of-Concept Implementierung ist. Ebenso vereinfachen sie die Implementierung an sich und machen diese einfacher zu verstehen. Einige der Punkte sind auch relevant für Sicherheitsbedenken. Dies betrifft vor allem die Programmiersprache und die Komplexität der inneren Logik.

Zur näheren Auswahl kamen aufgrund der genannten Gesichtspunkte folgende SMTP-Softwarelösungen:

1. Sendmail [45]

2. OpenSMTPD [36]

3. Haraka [53]

Sendmail kam in die nähere Auswahl, da es eine sehr alte Lösung ist, die extrem hohe Verbreitung hat. Ebenso ist der Code in weiten Teilen sehr gut dokumentiert. Allerdings ist die Konfigurationskomplexität äußerst hoch und der Code beinhaltet viel Indirektion, da der Code über die Zeit gewachsen ist. Sendmail ist in C geschrieben.

Als Alternative zu Sendmail fiel der Blick auf OpenSMTPD, dessen erstes stabiles Release aus dem ersten Quartal 2013 stammt [35]. Dies versprach eine bessere Strukturierung des Codes als Sendmail. Allerdings musste festgestellt werden, dass so gut wie keine Dokumentation auf Code-Ebene vorhanden ist. OpenSMTPD ist ebenfalls in C geschrieben. Aufgrund der mangelhaften Dokumentation schied dies also Option aus.

Als weitere Alternative wurde Haraka betrachtet. Haraka ist ein SMTP Server, welcher in Node.js geschrieben ist, also praktisch in Javascript. Dies öffnete einige Bedenken, da Javascript schwach und dynamisch typisiert ist und diverse Details des SMTP-Protokolls auf sehr exaktem Verhalten bezüglich der Bit-Länge gesendeter Nachrichten beruhen. Allerdings besitzt Haraka ein Plugin-System, welches auf Callbacks bzw. Hooks basiert, die es dem Programmierer erlauben, an unterschiedlichen Stellen in der Verarbeitung einer Transaktion anzusetzen und diese zu überschreiben oder lediglich zusätzliche Logik zu definieren. So ist es beispielsweise möglich, zu definieren was der Haraka Server nach einem empfangenen `DATA` SMTP-Kommando tut, ohne bereits bestehenden Code verändern zu müssen. Ebenso ist die Dokumentation in weiten Teilen ausreichend, weshalb diese SMTP-Softwarelösung trotz der Bedenken über die Schwächen des Javascript-Typsystems als Basis der Implementierung ausgewählt wurde.

Übersicht über Haraka

Wie bereits erwähnt, basiert Haraka auf einem explizit entwickelten Plugin-System. Weite Teile der Kernfunktionalität von Haraka sind über das Plugin-System selbst implementiert und werden als Standard-Plugins mitgeliefert. Diese können dann über diverse Konfigurationsdateien im Ordner `config` direkt gesteuert werden. Ebenso kann definiert werden, welche Plugins überhaupt ausgeführt werden und in welcher Reihenfolge. Dies ist der Kern der Haraka-Konfiguration und befindet sich in der Datei `config/plugins` im Basisordner.

Interessante hooks, die für die Implementierung von Bedeutung sind oder sein können, sind die folgenden:

- `connect_init`: wird beim Start einer Verbindung aufgerufen

- `lookup_rdns`: liefert den reverse DNS
- `connect`: wird nach `lookup_rdns` aufgerufen
- `mail`: hier können die Absender examiniert werden
- `rcpt`: hier können die Rezipienten examiniert werden
- `data`: wird beim `DATA` SMTP Kommando aufgerufen
- `data_post`: wird am Ende des `DATA` Kommandos aufgerufen, wenn der Client mit einem Punkt in einer Zeile signalisiert, dass die E-Mail Daten vollständig sind und der Server antworten kann
- `queue`: bevor die Mail in die Queue gelangt
- `get_mx`: wird beim Versenden aufgerufen, um den MX Record des empfangenden MTAs festzustellen

Das Plugin-Konzept sieht vor, dass mehrere Plugins denselben Hook verwenden können. Dafür muss am Ende der Funktion ein entsprechender Return Code zurückgegeben werden.

Um ein Open Relay zu definieren, kann man beispielsweise beim `rcpt` Hook ansetzen, und ,abhängig von diversen Regeln, eine Weiterleitung erlauben oder nicht:

```
1  /**
2   * This relays the mssage if is_valid_rcpt(recipient)
      returns true
3   */
4  exports.hook_rcpt = function(next, connection, params) {
5    // check if the recipient is valid (e.g. in a list)
6    if (is_valid_rcpt(params[0]) == true) {
7      connection.relaying = true;
8    }
9    return next(OK); // no further plugins on this hook will
      run
10 }
```

Code 4.4: Haraka Relay

Wie hier zu sehen ist, existieren in Haraka einige grundsätzliche Objekte, die etwaige Informationen tragen. Auf der äußersten API Schicht befindet sich das »Connection Object« (`connection` auf Code-Ebene). Dies wird für jede Verbindung, die Haraka mit einem anderen MTA eingeht erstellt und hat eine eindeutige UUID. Es trägt allgemeine Informationen über den Remote Server. Ebenso stellt es allerdings die Verbindung zur nächsten API-Schicht her, der Transaktion. `connection.transaction`

ist ein Objekt, das die gerade ablaufende Transaktion beschreibt. Es ist erst valide, nachdem `MAIL FROM` gesendet wurde und wird beim Erreichen der Mail-Queue vernichtet wenn `RSET` gesendet wird oder wenn `MAIL FROM` nicht akzeptiert wurde. Innerhalb des Transaktions-Objekts sind weitere Objekte gekapselt, die die Informationen über den eigentlichen Vorgang enthalten. Das Transaktions-Objekt ist im Grunde der Umschlag der E-Mail. Es enthält die SMTP Informationen über Empfänger in `transaction.rcpt_to` und Sender in `transaction.mail_from`, ebenso wie die E-Mail selbst über die Objekte `transaction.header` (alle Headerfelder) und `transaction.body` (Text der E-Mail). Hierüber ist es möglich, an alle Informationen während einer SMTP-Sitzung zu gelangen, diese zu manipulieren oder abhängig von den Daten (z.B. Existenz von Headerfeldern) weitere Logik ausführen zu lassen. Weiterhin existiert ein ein »Outbound Modul«, welches über das Objekt `outobund` ansprechbar ist. Über dieses Objekt ist es möglich, explizit E-Mails zu verschicken, beispielsweise wenn ein Plugin während der Verarbeitung selbst eine neue E-Mail schicken möchte. Als letztes interessantes Objekt existiert im Connection Objekt das Objekt `connection.results`, welches dazu benutzt werden kann Zustände und Informationen zu speichern, die dann über mehrere Plugins hinaus abrufbar sind (z.B. logs).

Wahl des Einsprungpunktes

Für die Wahl des Einsprungpunktes kamen mehrere Ansätze in die nähere Auswahl.

Der erste Ansatz war die E-Mails kurz bevor sie die Queue erreichen, im Hook `queue` abzufangen und die E-Mail manuell weiterzuleiten, beispielsweise indem über `outbound.send_email(...)` eine neue E-Mail erstellt und versendet wird. Das Problem hierbei ist allerdings, dass der `queue` Hook speziell ist und man diesen nur vollständig überschreiben kann, nicht aber nur zusätzliche Logik einführen kann. Dies erhöht den Implementierungsaufwand.

Eine weitere Möglichkeit wäre gewesen, an mehreren Stellen anzusetzen, beispielsweise bei den Hooks `mail` und `rcpt`, und für jede Station Informationen zu sammeln, diese dann im `connection.results` Objekt zu speichern und dann zu einem späteren Zeitpunkt eine neue E-Mail zu versenden oder die bestehende zu manipulieren. Da dies allerdings erhöhter Aufwand ist und einen globalen Zustand beinhaltet, wurde diese Idee verworfen.

Der letzte Ansatz war, im Hook `data_post` anzusetzen, direkt nachdem die E-Mail Daten eingegangen sind, also das SMTP Kommando `DATA` abgeschlossen ist. Dies hat mehrere Vorteile. Zum einen kann dieser Hook durch andere Plugins normal weiterverarbeitet werden und zum anderen erlaubt er an dieser Stelle das Transaktionsobjekt, welches gerade erstellt wurde, ohne viel Seiteneffekte direkt zu manipulieren. Ebenso

ist dies eine zentrale Stelle, an der noch relativ wenig Plugin-Logik ausgeführt worden ist, sodass kein kompliziertes Speichern von Zwischendaten erforderlich ist.

Somit fiel die Wahl auf den letzten Ansatz und hatte folgende Implikationen:

- Einsprungpunkt nachdem das SMTP Kommando `DATA` vollständig ausgeführt wurde (nach dem E-Mail Text)
- keine Zwischenzustände nötig
- erlaubt das Weiterverarbeiten durch andere Plugins
- direktes Modifizieren des Transaktionsobjektes: E-Mail und andere Meta-Daten werden direkt manipuliert

Implementierung MTA-Entscheider

Nachfolgend wird der Hauptteil des Implementierungscodes aufgezeigt, allerdings ohne die Hilfsfunktionen. Dies soll eine Übersicht über den Haraka-spezifischen Algorithmus für den MTA-Entscheider geben.

```
exports.hook_data_post = function(next, connection) {
    // if we have a myst mail, we need to inject our
   Protocol
    if (is_myst_mail(connection.transaction.header)) {
        // we decrypt the mail body and treat it as raw
        // mail data
        var decrypted_email = myst_decrypt(connection.
          transaction.body.bodytext);

        // replace the current transaction object
        var t = transform_transaction(decrypted_email,
          connection);
        connection.transaction = t;

        // the unpacked mail is a myst mail too,
        // relay it
        if (is_myst_mail(connection.transaction.header)) {
            connection.relaying = true;
        }
    }

    // let the other handlers do what they want, we are
    // finished injecting our logic, the rest is still
    // SMTP compliant
    next();
}
```

Code 4.5: Haraka MTA-Entscheider

Die Funktionen `myst_decrypt()` und `transform_transaction()` können zunächst leere Funktionen sein. Wichtig ist hier lediglich die Funktion `is_myst_mail()`, welche folgendermaßen definiert ist:

```
function is_myst_mail(header) {
    if (header.get("X-Myst")) {
        return true;
    }

    return false;
}
```

Code 4.6: Haraka Myst header checker

Damit ist der erste Teil des MTA-Entscheider bereits implementiert und die Logik für das Entpacken und Erstellen der neuen E-Mail kann implementiert werden.

Die Verschlüsselung und Entschlüsselung ist hier lediglich Base64. Die Funktion `transform_transaction()` wird aufgrund ihrer Komplexität nur im Anhang gezeigt, da sie ebenso auf interner Haraka-API aufbaut.

Ebenso wichtig ist, anzumerken, dass `connection.relaying = true;` im MTA-Entscheider bedeutet, dass der MTA für alle MystMails ein Open Relay ist. Deshalb kann und sollte ab Zeile 16 weitere Logik darüber entscheiden (z.B. eine Whitelist), ob die E-Mail weitergeleitet werden darf.

SMTP-Gateway und MystMail Erstellung

Um die initiale MystMail zu erstellen, die die Informationen über die Hops verschlüsselt in sich trägt, wird diese der Einfachheit halber im MSA, also gewissermaßen einem SMTP-Gateway, generiert.

Die Liste der Hops ist statisch und vordefiniert für Testzwecke. Es liegt somit kein Zufallsalgorithmus vor. Die Verschlüsselung ist ebenso lediglich Pseudoverschlüsselung und kodiert gemäß Base64.

```
function create_myst_mail(transaction) {
  // recipient is always the last hop
  var recipient = transaction.rcpt_to[0].host;
  var hops = get_hops();
  hops.push(recipient);

  // innermost mail, untouched
  var old_mail = get_raw_mail(transaction.body);

  var i;
  var new_mail = old_mail;
  for (i = hops.length - 1; i >= 0; i--) {
    if (i == 0) {
      // first hop, "from" must be the actual sender
      var sender = transaction.mail_from.host;
      new_mail = create_mail_for_hop(sender, hops[i],
                  date, new_mail);
    } else {
      new_mail = create_mail_for_hop(hops[i-1],
                  hops[i], date, new_mail);
    }
  }

  return new_mail;
}
```

Code 4.7: Haraka Myst Erstellung

Die Hilfsfunktionen lassen sich im Anhang einsehen.

Nachdem die Mail erstellt ist, wird ähnlich wie beim MTA-Entscheider im Hook `data_post` angesetzt und das Transaktions-Objekt manipuliert, um die dort enthaltene E-Mail mit der MystMail auszutauschen. Der entsprechende Code ist etwas kompliziert und ebenfalls im Anhang einsehbar.

Haraka Konfiguration

Nun müssen die Implementierungen noch adäquat in das Plugin-System integriert werden. Haraka Plugins werden erst ausgeführt, wenn sie in der Konfigurationsdatei `config/plugins` in der korrekten Reihenfolge aufgelistet sind. Ebenso müssen einige weitere Anpassungen gemacht werden, damit ein Testlauf möglich wird. Beispielsweise muss der `get_mx` Hook überschrieben werden, da hier nur lokale Systeme ohne MX-Record vorliegen. Die Implementierung ist ebenso im Anhang einsehbar.

Die Konfigurationsdatei für die Hops ist folgendermaßen definiert:

```
1 my_mx
2 access
3 dnsbl
4 helo.checks
5 open_relay
6 data.headers
7 enable_parse_body
8 my_transaction
9 max_unrecognized_commands
```

Code 4.8: Hop-Konfiguration

Die Konfigurationsdatei für das SMTP-Gateway ist folgendermaßen definiert:

```
1 my_mx
2 access
3 dnsbl
4 helo.checks
5 mail_from.is_resolvable
6 open_relay
7 data.headers
8 enable_parse_body
9 create_myst_mail
10 max_unrecognized_commands
```

Code 4.9: MSA-Konfiguration

Der Hauptunterschied liegt lediglich in Zeile 8 und Zeile 9, welche die Plugins beschreiben, die über `data_post` das Transaktions-Objekt verändern.

Kapitel 5

Evaluation & Validation

Das Testsystem wird gemäß des im Anhang befindlichen Handbuchs gestartet. Bei jedem Hop wird über die Konsole überwacht, welche Informationen eingehen. Ebenso werden temporär eingehende E-Mails in eine Datei geschrieben, um diese debuggen zu können. Der Testlauf zeigt auf:

- die E-Mail erreicht den Empfänger Container
- die eingegangene E-Mail am Empfänger Container stimmt mit der abgesendeten E-Mail überein
- in jedem Hop ist nur die MystMail sichtbar, welche im Body in Base64 Kodierung die nächste MystMail beinhaltet

Da die Implementierung allerdings nur diverse Teile des Algorithmus implementiert, erfüllt sie auch nicht exakt die Anforderungen aus Abschnitt 2.1. Dinge, die erfolgreich implementiert wurden, sind:

- MystMail MTA Entscheider als Open Relay (basierend auf dem `X-Myst-Mail` Headerfeld)
- MystMail Erstellung auf SMTP-Gateway Ebene mit statischer Hopliste und Base64 Kodierung anstatt kryptografischer Verschlüsselung

Damit kann allerdings bereits nachgewiesen werden, dass der Algorithmus im Kern funktioniert und nur die Informationen an die Hops gelangen, die für diese sichtbar sein sollen. Auf IMF Ebene sind die Informationen durch die Kodierung im Body geschützt und auf IP-Ebene durch das zufällige Routing.

Es kann allerdings nicht nachgewiesen werden, dass eine in der Praxis nutzbare Implementierung möglich oder sinnvoll ist, da u.a. wesentliche Teile der funktionalen Anforderungen aus Abschnitt 2.1 fehlen wie z.B. tatsächliche kryptografische Verschlüsselung oder ein Algorithmus der eine zufällige Route auswählt. Für einen Nachweis der praktischen Tauglichkeit müssten weitaus mehr Kriterien betrachtet werden

und die Implementierung einen anderen Umfang haben. Auf Seite der nichtfunktionalen Anforderungen aus Abschnitt 2.2 wurde zumindest NFA 1 eingehalten, obschon hier der Umstand ausgelassen wird, dass dennoch ein Ökosystem offener E-Mail Relay Server existieren muss, welche den hier beschriebenen Algorithmus beispielsweise als Plugin bereitstellen müssen. Eine MystMail kann nicht über gewöhnliche SMTP Server weitergeleitet werden, da für die Weiterleitung spezielles Wissen erforderlich ist.

Dennoch ist das Kernziel erreicht, indem das Weiterleiten und das Ver -und Entpacken von MystMails erfolgreich implementiert wurde.

Kapitel 6

Ausblick

Da nun die Implementierung durchgeführt wurde und dieselbe anhand der definierten funktionalen und nichtfunktionalen Anforderungen validiert wurde, müssen nachfolgend vor allem die Teile betrachtet werden, die nicht oder nicht ausreichend gelöst werden konnten, sei es weil eine adäquate Lösung den Rahmen dieser Arbeit übersteigt oder weil noch keine vorliegt. Demnach sind sowohl theoretische als auch praktische Probleme hier von Belang.

Diese werden im folgenden ausgeführt und beschreiben im Ganzen noch fehlende Lösungen für das vorgeschlagene System.

6.1 Theoretische Probleme

Zu den theoretischen Problemen gehören vor allem die mathematisch und kryptografisch relevanten Details des Algorithmus und der Implementierung.

Verschlüsselungsalgorithmus

Für die Verschlüsselung der MystMail selbst, also die Verschachtelung im Body der E-Mail, ist hier keine Spezifikation angegeben. Damit allerdings unterschiedliche E-Mail Server miteinander kommunizieren können, muss es entweder einen fest definierten Algorithmus geben oder es müssen durch genau spezifizierte Informationsweitergabe unterschiedliche Algorithmen unterstützt werden können. Im letzteren Fall würde dies die Fehlerbehandlung verkomplizieren, wenn ein empfangender E-Mail Server den Algorithmus zum Entschlüsseln der MystMail nicht beherrscht.

Ebenso muss das Verschlüsselungssystem adäquat mit einem Public-Key System verknüpft sein.

Public-Key System

Neben der Wahl des Verschlüsselungsalgorithmus ist im selben Kontext das Problem eines Public-Key Systems äußerst relevant.

Dies beinhaltet mehrere Punkte. Zum einen muss noch spezifiziert werden, welches Schlüsselsystem (ein Paar aus öffentlichem und privatem Schlüssel) überhaupt verwendet wird. Weiterhin muss festgelegt werden, in welcher Weise die öffentlichen Schlüssel verbreitet werden. Dies könnte beispielsweise ähnlich wie bei DKIM [10] erfolgen, welches die öffentlichen Schlüssel im DNS System hinterlegt (vgl. [10, S. 6]). Diese können dann automatisch abgerufen werden. Zusätzlich kann DNSSEC [3] verwendet werden, um das System insgesamt robuster zu machen. Denkbar ist allerdings auch ein Schlüsselsystem mit einer zentralen Autorität.

Routing-Algorithmus

Für das Routing selbst wird ein Zufallsalgorithmus benötigt. Dieser sollte allerdings so spezifiziert werden, dass er es vermeidet, bereits ausgewählte Routen wiederzuverwenden. Ebenso muss der Grad der Entropie sichergestellt werden, beispielsweise durch eigens entwickelte Algorithmen und Verwendung mehrerer Entropie-Quellen.

Die Anzahl der Knoten, die mindestens benötigt werden, um eine adäquate Anonymisierung zu erreichen, muss ebenso festgelegt werden. Dies kann auch praktische Untersuchungen erfordern.

Hier könnten Details des Onion routing [47] [12] [22] [13] untersucht werden, welches vom Konzept her ähnlich ist.

Kryptografie-System insgesamt

Weiterhin muss das Kryptografiesystem insgesamt betrachtet werden und Schwachstellen und Fehler untersucht werden, die bereits auf theoretischer Ebene ausgemacht werden können. Hierbei können u.U. auch ältere Arbeiten wie »Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms« [7] von Nutzen sein. Diese Arbeit untersucht ein Kryptografiesystem zur sicheren E-Mail Kommunikation auf abstrakter Ebene und macht keine konkreten Aussagen über benutzte Protokolle.

Fehlen einer Spezifikation

Das Fehlen einer konkreten Spezifikation, bzw. einer Protokollerweiterung ist eines der Hauptprobleme dieser Arbeit, welches auch einige der vorher genannten Probleme mit einschließt. Erst wenn eine vollständige Spezifikation, beispielsweise in Form eines RFC, vorliegt, kann auf konkreter Ebene über Implementierungsprobleme, Logikfehler und Durchführbarkeit diskutiert werden.

Nach jetzigem Stand ist es lediglich möglich, über die Idee und algorithmischen Einzelteile einer möglichen Lösung zu diskutieren.

6.2 Praktische Probleme

Zu den praktischen Problemen gehören vor allem Implementierungsprobleme und diverse Probleme, die sich aus einer praktischen Benutzung des vorgeschlagenen Systems ergeben.

Professionelle Implementierung

Nicht gerade ein Problem dieser Arbeit, aber ein Problem für eine mögliche praktische Benutzung ist das Fehlen einer professionellen Implementierung. Diese ist aber nicht sinnvoll ohne eine konkrete Spezifikation wie auf Seite 59 genannt möglich.

Die hier präsentierte Implementierung ist eine Proof-of-Concept Implementierung und in keiner Weise für den Einsatz gedacht. Unschöner Weise hängt sie sogar von interner API ab. Eine professionelle Implementierung würde hingegen als strenges Modul bzw. Plugin entwickelt werden müssen. Darüber hinaus müssen weitere wichtige Gesichtspunkte berücksichtigt werden wie die Wahl der zugrundeliegenden SMTP Implementierung und Sprache.

Dieser Punkt beinhaltet implizit einige der nachfolgend noch genannten Probleme.

Fehlerbehandlung

Die Fehlerbehandlung ist noch nicht ausreichend gelöst. Sie ist ebenso relevant für die Betrachtung der Sicherheit des Systems.

Dies ist auch teilweise ein theoretisches Problem, aber vor allem ein Problem auf Implementierungsebene. So ist es nicht klar, wie die Knoten sich verhalten sollen, wenn es Probleme bei der Weiterleitung, Entschlüsselung oder Verbindung gibt. Diese Probleme müssen robust gelöst werden, um die Angriffsfläche zu minimieren.

Ebenso denkbar sind Angriffe durch den Myst-Header, wenn beispielsweise missgebildete E-Mails versendet werden, die nicht korrekt oder gar nicht verschlüsselt sind und eventuell undefiniertes Verhalten im MTA hervorrufen.

Verifizierung und Authentizität

Obwohl es hier primär um das Unterdrücken und Verschleiern von Informationen geht, ist in praktischen E-Mail Systemen das Verifizieren der Authentizität von Verbindungen im Kontext des SMTP-Protokolls von hoher Bedeutung und das aus einer Vielzahl von Gründen.

Zum einen existiert, wie bereits erwähnt, das Problem der deformierten E-Mails und wie mit diesen umgegangen wird. Ebenso ist es möglich, auf IMF Ebene praktisch alle Headerfelder vorzutäuschen bzw. zu imitieren. Eine Teillösung hierfür wäre, alle Headerfelder mit DKIM [10] zu schützen, was bedeutet, dass selbst ein MITM diese nicht verändern könnte, da die Verifikation beim empfangenden MTA fehlschlagen würde. Damit könnte auch das `X-Myst-Mail` Headerfeld geschützt werden und es wäre nicht möglich, diesen zu entfernen.

Dabei muss allerdings untersucht werden, welche Auswirkungen die Nutzung von DKIM auf das Kryptografiesystem insgesamt hat und ob dies zu Informationsflüssen führen kann, die die Anonymität der kommunizierenden Parteien kompromittieren kann.

Spam-Abwehr und Traffic

Wie für jedes E-Mail System stellt sich die Frage der Spamabwehr und ob das vorgeschlagene System diese erschwert. Theoretisch sind zumindest zwischen den Knoten auf der Anonymisierungs-Route alle gewöhnlichen Mechanismen zur Spamabwehr möglich. Allerdings liegt das Problem weniger auf Ebene der Knoten. Auch wenn diese E-Mails zurückweisen können, die von nicht authentischen Absendern sind oder diverse Formvorgaben an eine MystMail nicht einhalten, können sie jedoch praktisch keine Aussage darüber machen, ob die E-Mail, die für den schlussendlichen Empfänger bestimmt ist, Spam ist. Denn diese kann nur der Empfänger entschlüsseln und einsehen.

Demnach kann eine Spam-Mail, die trotzdem eine korrekte MystMail ist, zwar beim Empfänger als solche erkannt werden, durchläuft aber dennoch die gesamte Anonymisierungs-Route und verursacht Traffic. Es muss untersucht werden, wie hoch das Potenzial einer Spam-Attacke ist, die DDoS-artige Auswüchse hat und eventuell das System als ganzes lahm legen könnte. Im schlimmsten Fall könnten diese Probleme sogar zum Blacklisting eigentlich unschuldiger Endknoten führen, die ohne ihr Wissen Spam weiterleiten, da sie gewissermaßen als Open Relays fungieren.

Zumindest ist es denkbar, eine strikte Authentifizierung zwischen den Knoten zu etablieren, damit valide Knoten bekannt sind und verifiziert werden können. Dies würde auch teilweise das Problem von böswilligen Knoten lösen, die möglicherweise E-Mails nicht weiterleiten. Dafür ist ein sogenannter Directory Server von Nöten.

Directory Server

Sowohl zur Verbesserung der Spam-Abwehr als auch zur Integrität des Systems insgesamt ist ein Directory Server notwendig. Dieser muss alle validen Knoten für eine mögliche Anonymisierungs-Route verwalten. Sowohl beim Erstellen einer Route wird

diese Liste abgefragt als auch bei der Kommunikation zwischen Knoten. Somit können Knoten mit Fehlverhalten auch problemlos entfernt werden.

Allerdings öffnet dies einen zentralen Angriffspunkt auf das System. Der Directory Server muss mit speziellen Methoden gesichert werden und die abgerufene Liste kryptografisch verifizierbar sein.

Zuverlässigkeit

Ein E-Mail System ist nur nutzbar und praktikabel, wenn es einen hohen Grad an Zuverlässigkeit hat. Sowohl die Probleme der Fehlerbehandlung als auch die Verschlüsselung, das Routing und diverse Probleme bei der Spam-Abwehr können die Zuverlässigkeit des Systems beeinträchtigen.

Deshalb muss eine Protokollerweiterung bzw. ein Mechanismus entwickelt werden, der es erlaubt, ohne Verlust von Anonymität, das Erreichen einer E-Mail beim Empfänger bestätigen zu lassen. Denkbar wäre nach einem bestimmten Intervall auf eine Bestätigungsmail, welche über eine neue zufällige Route versendet wird, zu warten. Hier stellt sich allerdings die Frage, ob über die Intervalle statistische Analysen möglich sind. Ebenso kann die Bestätigungsmail fehlschlagen und nicht ankommen.

Gesamtsystem

Das hier vorgeschlagene System ist nicht nur eine SMTP-Erweiterung. Es handelt sich vor allem um eine Kombination aus bereits existierenden Technologien wie GPG, TorBirdy und TLS.

Selbst wenn für die SMTP-Erweiterung eine konkrete Spezifikation und eine funktionierende Implementierung existiert, muss der Benutzer dennoch weitere Schritte unternehmen, um dieses System korrekt zu benutzen. Dies erhöht die Wahrscheinlichkeit, dass das System inkorrekt benutzt wird.

Dies ist einerseits Dokumentationsarbeit, andererseits ist es aber auch denkbar, ähnlich wie Tor, eine Softwarelösung zur Verfügung zu stellen, die eine korrekte Konfiguration beinhaltet. Allerdings ist dies schwierig, da dies nur auf MUA Ebene möglich ist und der E-Mail Server immer noch fehlkonfiguriert sein kann.

Ebenso ist das Problem der SMTP Spezifikation ungelöst, welche keine Garantie gibt, dass 2 MTAs überhaupt verschlüsselt kommunizieren, selbst wenn beide dies beherrschen. Dies muss auf Implementierungsebene garantiert werden, für alle Komponenten, die in diesem Kryptografiesystem mitwirken.

Anhang A

Beiliegende Datenträger

Dieser Arbeit liegt eine CD bei. Nachfolgend wird der Inhalt aufgelistet und beschrieben.

A.1 Inhaltsliste

- CD 1
  - Thesis
    - Binary
    - Source
  - Webseiten
  - RFCs in Textform
  - Implementierung
    - Dockerfiles
    - Source
    - Konfigurationsdateien
  - Dokumentation
    - Handbuch

A.2 Inhaltsbeschreibung

Die Implementierung befindet sich im Unterordner `Implementierung` und beinhaltet dort auch den Quellcode für die Docker Images `alpine-haraka`, `alpine-haraka-gateway` und `alpine-swaks`. Die Haraka Plugins sind also jeweils in den Ordnern `Implementierung/alpine-haraka/plugins` und `Implementierung/alpine-haraka-gateway/plugins`.

Der Latex Quellcode befindet sich im Unterordner `LaTeX` und ein Symlink zur PDF existiert im Root-Ordner als `Bachelorthesis.pdf`.

Alle RFCs sind in Textform im Unterordner `RFCs` aufzufinden. Ebenso sind alle referenzierten Webseiten statisch im Unterordner `Webseiten` verfügbar.

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Abkürzungsverzeichnis

bzw.	beziehungsweise
ca.	circa
DDoS	distributed denial-of-service
d.h.	das heißt
E-Mail	electronic mail
FTP	File Transfer Protocol
FQDN	Fully qualified domain name
engl.	englisch
etc.	et cetera
ID	Identifikationsnummer
IMAP4	Internet Message Access Protocol Version 4
IMF	Internet Message Format
Mrd.	Milliarden
POP3	Post Office Protocol Version 3
RFC	Request for Comments
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
u.a.	unter anderem
UUID	Universally Unique Identifier
u.U.	unter Umständen
WWW	World Wide Web
z.B.	zum Beispiel

Literaturverzeichnis

[1] Allen, Christopher: *Regularizing Port Numbers for SSL*. https://lists.w3.org/Archives/Public/ietf-tls/1997JanMar/0079.html, 1 1999. [Online; geprüft 10.03.16].

[2] Alliance, Dark Mail: *The DIME resolver library and command line utilities*. https://github.com/lavabit/libdime, 02 2016. [Online; geprüft 14.03.16].

[3] Arends, R., R. Austein, M. Larson, D. Massey und S. Rose: *Protocol Modifications for the DNS Security Extensions*. RFC 4035 (Proposed Standard), März 2005. Updated by RFCs 4470, 6014, 6840.

[4] Berjon, Robin, Steve Faulkner, Travis Leithead, Silvia Pfeiffer, Edward O'Connor und Erika Doyle Navara: *HTML5*. Candidate Recommendation, W3C, Juli 2014. http://www.w3.org/TR/2014/CR-html5-20140731/.

[5] Borenstein, N. und N. Freed: *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. RFC 1521 (Draft Standard), September 1993. Obsoleted by RFCs 2045, 2046, 2047, 2048, 2049, updated by RFC 1590.

[6] Callas, J., L. Donnerhacke, H. Finney, D. Shaw und R. Thayer: *OpenPGP Message Format*. RFC 4880 (Proposed Standard), November 2007. Updated by RFC 5581.

[7] Chaum, David L.: *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms*. Commun. ACM, 24(2):84–90, Februar 1981.

[8] Crispin, M.: *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard), März 2003. Updated by RFCs 4466, 4469, 4551, 5032, 5182, 5738, 6186, 6858.

[9] Crocker, D.: *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. RFC 822 (INTERNET STANDARD), August 1982. Obsoleted by RFC 2822, updated by RFCs 1123, 2156, 1327, 1138, 1148.

[10] Crocker, D., T. Hansen und M. Kucherawy: *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376 (INTERNET STANDARD), September 2011.

[11] Dierks, T. und E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685.

[12] Dingledine, Roger, Nick Mathewson und Paul Syverson: *Tor: The Second-generation Onion Router*. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, Seiten 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[13] Dingledine, Roger, Nick Mathewson und Paul Syverson: *Tor: The Second-Generation Onion Router*. In: *IN PROCEEDINGS OF THE 13 TH USENIX SECURITY SYMPOSIUM*, 2004.

[14] Docker: *Docker Homepage*. `https://www.docker.com/`, 02 2016. [Online; geprüft 14.03.16].

[15] Ferguson, Niels und Bruce Schneier: *Practical Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 1 Auflage, 2003.

[16] Freed, N. und N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. RFC 2049 (Draft Standard), November 1996.

[17] Freed, N. und N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045 (Draft Standard), November 1996. Updated by RFCs 2184, 2231, 5335, 6532.

[18] Freed, N. und N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046 (Draft Standard), November 1996. Updated by RFCs 2646, 3798, 5147, 6657.

[19] Freed, N. und J. Klensin: *Media Type Specifications and Registration Procedures*. RFC 4288 (Best Current Practice), Dezember 2005. Obsoleted by RFC 6838.

[20] Freed, N. und J. Klensin: *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC 4289 (Best Current Practice), Dezember 2005.

[21] Gellens, R. und J. Klensin: *Message Submission for Mail*. RFC 6409 (INTERNET STANDARD), November 2011.

[22] Goldschlag, David, Michael Reed und Paul Syverson: *Onion Routing for Anonymous and Private Internet Connections*. Communications of the ACM, 42:39–41, 1999.

[23] Hoffman, P.: *SMTP Service Extension for Secure SMTP over TLS*. RFC 2487 (Proposed Standard), Januar 1999. Obsoleted by RFC 3207.

[24] Hoffman, P.: *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207 (Proposed Standard), Februar 2002.

[25] Klensin, J.: *Simple Mail Transfer Protocol*. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336.

[26] Klensin, J.: *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard), Oktober 2008. Updated by RFC 7504.

[27] Levison, Ladar: *Dark Internet Mail Environment: Architecture and Specifications*. `https://darkmail.info/downloads/dark-internet-mail-environment-march-2015.pdf`, 03 2015. [Online; geprüft 14.03.16].

[28] Luqi: *Software evolution through rapid prototyping*. Computer, 22(5):13–25, May 1989.

[29] Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text*. RFC 1522 (Draft Standard), September 1993. Obsoleted by RFCs 2045, 2046, 2047, 2048, 2049.

[30] Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. RFC 2047 (Draft Standard), November 1996. Updated by RFCs 2184, 2231.

[31] Mozilla: *Thunderbird Homepage*. `https://www.mozilla.org/en-US/thunderbird/`, 2 2016. [Online; geprüft 14.03.16].

[32] Myers, J. und M. Rose: *Post Office Protocol - Version 3*. RFC 1939 (INTERNET STANDARD), Mai 1996. Updated by RFCs 1957, 2449, 6186.

[33] Nakamoto, Satoshi: *Bitcoin: A Peer-to-Peer Electronic Cash System*. Technischer Bericht, Bitcoin, 2009.

[34] Newman, C.: *Using TLS with IMAP, POP3 and ACAP*. RFC 2595 (Proposed Standard), Juni 1999. Updated by RFC 4616.

[35] OpenBSD: *OpenSMTPD first release Website*. `http://www.openbsd.org/53.html`, 02 2016. [Online; geprüft 14.03.16].

[36] OpenBSD: *OpenSMTPD Homepage*. https://www.opensmtpd.org/, 02 2016. [Online; geprüft 14.03.16].

[37] Partridge, C.: *Mail routing and the domain system*. RFC 974 (Historic), Januar 1986. Obsoleted by RFC 2821.

[38] Popper, K.R. und H. Keuth: *Karl Popper: Logik Der Forschung*. Klassiker Auslegen. Akademie Verlag GmbH, 1998.

[39] Postel, J.: *Simple Mail Transfer Protocol*. RFC 821 (INTERNET STANDARD), August 1982. Obsoleted by RFC 2821.

[40] Resnick, P.: *Internet Message Format*. RFC 2822 (Proposed Standard), April 2001. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.

[41] Resnick, P.: *Internet Message Format*. RFC 5322 (Draft Standard), Oktober 2008. Updated by RFC 6854.

[42] Saltzmann, Roi und Adi Sharabani: *Active Man in the Middle Attacks*. Technischer Bericht, IBM Rational Application Security Group, 02 2009.

[43] Sara Radicati, PhD und Justin Levenstein: *Email Statistics Report, 2013-2017*. http://www.radicati.com/wp/wp-content/uploads/2013/04/Email-Statistics-Report-2013-2017-Executive-Summary.pdf, 4 2013. [Online; geprüft 14.03.16].

[44] Sasaki, Yu, Lei Wang, Kazuo Ohta und Noboru Kunihiro: *Security of MD5 Challenge and Response: Extension of APOP Password Recovery Attack*. In: Malkin, Tal (Herausgeber): *Topics in Cryptology – CT-RSA 2008*, Band 4964 der Reihe *Lecture Notes in Computer Science*, Seiten 1–18. Springer Berlin Heidelberg, 2008.

[45] Sendmail, Inc.: *Sendmail Homepage*. http://www.sendmail.com/sm/open_source/, 02 2016. [Online; geprüft 14.03.16].

[46] Sheffer, Y., R. Holz und P. Saint-Andre: *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. RFC 7457 (Informational), Februar 2015.

[47] Syverson, Paul F., David M. Goldschlag und Michael G. Reed: *Anonymous connections and onion routing*. In: *In IEEE Symposium on Security and Privacy*, Seiten 44–54, 1997.

[48] The GNU Privacy Guard Team: *The GNU Privacy Guard Homepage*. https://www.gnupg.org, 12 2015. [Online; geprüft 14.03.16].

[49] The Tor project: *Towards a Tor-safe Mozilla Thunderbird*. https://trac.torproject.org/projects/tor/raw-attachment/wiki/doc/TorifyHOWTO/EMail/Thunderbird/Thunderbird%2BTor.pdf, 7 2011. [Online; geprüft 14.03.16].

[50] The Tor project: *Tor Browser Homepage*. https://www.torproject.org/projects/torbrowser.html.en, 7 2015. [Online; geprüft 14.03.16].

[51] The Tor project: *TorBirdy Homepage*. https://trac.torproject.org/projects/tor/wiki/torbirdy, 7 2015. [Online; geprüft 14.03.16].

[52] Troost, R., S. Dorner und K. Moore: *Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field*. RFC 2183 (Proposed Standard), August 1997. Updated by RFCs 2184, 2231.

[53] Unbekannt: *Haraka Homepage*. https://haraka.github.io/, 02 2016. [Online; geprüft 14.03.16].

[54] Warren, Jonathan: *Bitmessage: A Peer-to-Peer Message Authentication and Delivery System*. Technischer Bericht, Bitmessage, New York, NY, USA, November 2012.

[55] Watson, R.W.: *Mail Box Protocol*. RFC 196, Juli 1971. Obsoleted by RFC 221.