Simon Roth, BSc.

An Optimistic Certified Email System Based Upon Internet Email

Master's Thesis

Graz University of Technology

Institute of Applied Information Processing and Communications
Head: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Supervisor: Dipl-Ing. Dr.techn. Arne Tauber

Graz, February 2018

Abstract

Traditional emails sent over the Internet lack in several points compared to registered postal letters. There is no reliable way to track emails, a sender has no reliable possibility to prove an email was sent, a sender receives no reliable confirmation when an email was delivered successfully and a recipient might deny a reception. This thesis focuses on the implementation of a certified email system covering those gaps whereby - in contrast to several solutions already in the field - the current Internet email technology is re-used as much as possible enhancing the end-user usability as well as simplifying the deployment on already available infrastructure. Furthermore, the selected certified email protocol is extended by a feature especially important for the business sector: Email forwarding while maintaining end-to-end encryption. This feature is possible by using proxy re-encryption - an emerging cryptographic primitive allowing proxies to re-encrypt ciphertext encrypted for one party to another party without getting access to the plaintext.

Kurzfassung

Herkömmliche Internet E-Mails haben mehrere entschiedene Nachteile gegenüber eingeschriebenen Briefen. So gibt es keine vertrauenswürdige Möglichkeit zur Nachverfolgung, der Sender hat keine vertrauenswürdige Möglichkeit zu beweisen, dass eine E-Mail versendet wurde, der Sender erhält keine vertrauenswürdige Bestätigung wenn eine E-Mail erfolgreich übermittelt wurde und der Empfänger kann den Empfang abstreiten. Diese Arbeit fokussiert auf die Implementierung eines zertifizierten E-Mail Systems welches diese Nachteile abdeckt. Dabei wird im Gegensatz zu anderen bereits existierenden zertifizierten E-Mail Systemen darauf geachtet, die bereits vorhandenen Internet E-Mail Technologien möglichst wieder zu verwenden um einerseits den Anwendern die Benutzung zu erleichtern und andererseits den Aufwand der Installation eines solchen Systems in Grenzen zu halten. Darüber hinaus wird das gewählte zertifizierte E-Mail Protokoll um eine Funktionalität erweitert, die speziell im Business Sektor von Bedeutung ist: E-Mail Weiterleitung unter Beibehaltung der Ende zu Ende Verschlüsselung. Diese Funktionalität wird durch den Einsatz von Proxy re-Encryption ermöglicht, eine aufstrebende Kryptografische Technologie die es Proxys ermöglicht verschlüsselten Text für eine Partie in verschlüsselten Text für eine andere Partie umzuwandeln ohne dabei Zugriff zum Originalen Inhalt zu bekommen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Acknowledgements

In first instance I want to thank my advisor Arne Tauber as well as my former colleague and advisor Andreas Fitzek for their patience, support and hints whenever needed. A very special thanks is due to my parents and grandmother for never stop supporting and believing in me. Nevertheless, the most important person for finishing this thesis is my wonderful girlfriend Tamara. Without all your motivation and support in every sense this would not have been possible.

Contents

List of Figures

1. Introduction

After highlighting the most important historical stages of development regarding message exchange from a very high level in section 1.1, the origination and evolution of email technology is investigated in section 1.2. In section 1.3 some statistics are provided pointing out the significance of emails today next to the most used sub-technologies in the field. The necessity of certified emails is described in section 1.4 followed by the motivation to provide a certified mail system based on the standardized email architecture in section 1.5. Finally, the methodology for implementing such a system is described in section 1.6.

1.1. A Short History of Message Exchange

The concept of exchanging messages between people over large distances is already thousands of years old and dates back to old cultures like the Babylonians or ancient Egyptians. The purpose of sending messages is still the same nowadays: a sender intends to pass information asynchronously to a distant recipient without traveling towards the receiving destination physically him or herself for transmission. Asynchronous communication hereby means that the sender may send the message independently of the recipient, hence it does not matter whether the addressee is able to receive or reply to the message simultaneously, in contrast to a telephone call for example. While different information carriers were used over time, like clay tablets or papyrus thousands of years ago or paper in more recent times on the other hand, they all had in common that it was necessary to forward the carrier physically, requiring a long period for transmission. Even tough paper is still a very common and broadly used carrier for messages, alternatives emerged with advances in technology. The key technology for raising

message exchange to a new level was the invention of electricity. Electricity first led to new transfer technologies like telegraphs or facsimile machines. The greatest progress achieved hereby was that it is no longer necessary to pass the messages physically and therefore they can be delivered in comparatively no time with less resources. The most important technology for information exchange widely used today came up with the invention of Internet and resulted in emails as successor over more or less outdated technologies like telegrams or facsimile.

1.2. The Rise of the Email

The invention of Internet email already started in the sixties, at a time when personal computers were not even imaginable and dates back to times in the mainframe computer era, long before the Internet itself came up. Usually multiple people shared a single computer at that time. Therefore, the launch of email in strict sense already began with single-computer electronic mails: according to researcher Raymond Tomlinson first programs like SNDMSG (Send Message) were developed making it possible for computer operators to send messages asynchronously to other operators using the same computer by employing simple files as mailboxes accessible only by the intended persons [1]. A little later the development of a predecessor of the modern Internet called ARPANET (Advanced Research Projects Agency Network) started by the end of the sixties whereas the intention of ARPANET was to establish a decentralized computer network for information exchange between researchers located at different universities. This led to first network protocols as well as network programs like an experimental file transfer program called CPYNET (Copy Net). 1971 RFC196 referred as "A Mail Box Protocol" was published suggesting a protocol for distributing messages to other destinations by sending them to remote printers [2]. This RFC inspired Tomlinson to integrate the program CPYNET into SNDMSG in order to allow sending messages via ARPANET to remote mailboxes [1] as he estimated sending messages to printers as too complicated. This was also the time when the now well-known at sign (@) was born in the email context, as an indicator for distinguishing between local and remote messages was necessary.

CPYNET was replaced by FTP (File Transfer Protocol) whereas the latter one was extended by mailing functionalities followed by several revisions updating the specification for further improvements [3]. It was determined that mail commands can only consist of ASCII (American Standard Code for Information Interchange) strings instead of binary data - a characteristic still valid today. More sophisticated UAs (User Agents) and MTAs (Mail Transfer Agents) were developed as the email handling was not satisfactory for an extensive usage. In 1973 the decision was made that email should get a separate protocol as it became too important and the community was shifting. The same year also led to first standards for email headers. The header standards were improved several times within the next years to meet the requirements before RFC-822 [4] was published in 1982, superseded by RFC-5322 [5] in 2008, the message format standard used today. In order to transfer emails from server to client applications in a standardized manner, the POP (Post Office Protocol) [6] and IMAP (Internet Message Access Protocol) [7] were introduced in the 80s whereas IMAP is a more sophisticated version compared to POP. Both protocols are still in wide use today.
As ARPANET was only accessible to limited institutions while the demand for computer networks increased in other places the same way, alternative networks like UUPC, BITNET and CSNET were developed whereas email applications similar to those originated in ARPANET were seen as an important key component [3]. Those new networks subsequently also got connected one below the other and to ARPANET resulting in the network we now call Internet. As parties began to develop their own variations of mailing applications, obviously the need for standardization became more important. Developing MTAs able to handle multiple networks transformed into a complex task and was faced with serious academic research resulting in the *Delivermail* (Eric Allman, UC Berkley), *mmdf* (Dave Crocker, University of Delaware) and *sendmail* (Eric Allman, UC Berkley) MTA superseding *Delivermail* - the latter still being one of the most used MTAs to date.
In 1980 the transition from Arpanet to Internet started requiring a bridge protocol between both networks. Suzanne Sluzier and Jon Postel proposed MTP (Mail Transfer Protocol). As this protocol turned out to be too complex Postel reworked MTP one year later resulting in a simpler protocol called SMTP (Simple Mail Transfer Protocol) - the protocol for transferring emails to date. *Sendmail* was the first MTA to implement SMTP (Simple Mail Transfer Protocol). In order to avoid the need of using numeric IP

numbers to address receiving destinations a distributed database called DNS (Domain Name System) was integrated in the mailing architecture using MX-RR (Mail Exchange Resource Records) for routing in 1986 [8]. The last important key component shaping the Internet email architecture today was released 1993 in RFC 1426 [9]: extending the SMTP protocol for sending MIME (Multipurpose Internet Mail Extensions) messages. MIME messages allow determination of encoding formats, to transmit binary data as attachments and to compose message bodies with multiple parts.

1.3. Email: State of the Art

The day when more emails were sent than traditional mails took place already decades ago: In 2016 worldwide approximately 215.3 billion emails per day were sent by 2.6 billion users whereas those numbers are still increasing. While electronic mails are very important for all forms of communication despite alternative technologies available, e.g., instant messengers or SMS, it is especially the leading technology for business communication [10].
Regarding protocols in the field, SMTP is the most used technology for sending respectively transporting emails nowadays. POP3 or IMAP as more sophisticated solution, is used for receiving emails by a client from the server. In 2017 *Exim*, *Postfix* and *Sendmail* are the most used server applications implementing SMTP according to a mail server survey [11]. For IMAP servers on the other hand *Dovecot* and *Courier* are the key players [12]. The distribution of mail clients changed heavily over the years: As emails usually were read and composed using desktop applications like *Microsoft Outlook* or *Thunderbird* some years ago those clients tend to be shifted more and more towards web applications like *gmail* or *outlook.com*. Additionally, email clients on mobile phones gain a considerable piece of the pie nowadays [13].

1.4. Certified Mail

Ordinary emails without further improvements or extending technologies can be compared to ordinary letters, or even worse considering traceability or confidentiality: postcards. There is no guarantee that a recipient will obtain a message - it simply might get lost during transmission. Regarding confidentiality an email akin to a postcard has to be treated as plain text message; every relaying station between sending and receiving is able to read the email for example. A recipient might deny the reception of the email on his will. Those specified attributes are only a subset of factors limiting the use of ordinary emails. It would be unwise to use regular emails for important business or governmental mails for instance. In the analogous world, i.e., letters in paper form, those problems are solved using registered mails. The realization of those registered mail services differs from country to country and provider to provider, but usually they have at least following characteristics or subsets of those characteristics (depending on the appropriate service) in common:

- A sender gets a receipt including the filing date proving the submission.
- Receivers have to sign an acknowledgment of receipt in order to get the registered mail handed over.
- This receipt including an original signature by the recipient is returned to the sender.
- Only persons intended to receive the letter may retain it.
- The recipient therefore has to prove his or her identity via some kind of identification card.
- Different stations of the letter during the delivery process can be tracked via tracking number.

1.5. Motivation

Sending letters analogously in paper form is not up to date anymore. It would be desired to offer a service making it possible to send digital certified emails while keeping all the characteristics provided by registered mail or

even extend them when reasonable and possible. Benefits in contrast to letters in paper form are for example a transmission in comparatively no time, fewer costs due to less personal costs and efforts for the physical delivery or reducing paper wastage. It is desired to integrate a solution for certified electronic emails in the already existing email infrastructure. Users will accept an integrated solution rather than new systems as they may use the applications they are familiar with and where they have already registered their accounts. An integrated solution can reuse several well tested components improved over time by gaining practical knowledge. Additionally it makes the administrators life simpler as they don't have to install and maintain new software stacks. As there exist several different implementations of IMAP, POP3 and SMTP server applications as well as clients it should be considered to provide some interfaces minimizing the changes to be made in the applications already existing in order to minimize the implementation effort and to obtain acceptance.

1.6. Methodology

After investigating the most important components of the current Internet email infrastructure, requirements requested by certified emails are analyzed. In a consequence certified email protocols introduced by the research community are examined as well as solutions already in the field. Based on those examination results the best fitting components and protocols are picked in order to implement a certified email system fulfilling several demands and complying with the current Internet email infrastructure as much as possible. Finally, the resulting system is analyzed regarding potential improvements.

2. Email Technology

In order to define requirements, to analyze systems already deployed in the field, to evaluate scientific papers proposing new ideas and to finally implement a protocol for certified emails extending the ordinary email architecture, it is necessary to investigate several important subcomponents and protocols of the email architecture commonly used today. Basic components of the mailing infrastructure are explained in section 2.1. Based upon this knowledge the SMTP protocol for transferring mails is described in section 2.2 next to POP3 in section 2.3 or IMAP as alternative to POP3 for receiving mails by user agents in section 2.4. The formats defining emails itself are described in section 2.5. Email security considerations are finally highlighted in section 2.6. Nevertheless, the following descriptions are on a rather high level; for low level specification details the appropriate RFC should be consulted.

2.1. Basics

Before going more into detail concerning the most important mailing protocols in use nowadays, an explanation of some basic components regarding the email infrastructure is below-mentioned. Those components are of vital importance to understand all protocols described within the next sections.

2.1.1. Mail Transfer Agent

The Mail Transfer Agent (MTA) is a component responsible for the mail transport itself [14]. It receives messages from various sources like Message Submission Agents or other MTAs. Usually a source has to be authenticated

by a MTA before accepting emails to be transmitted. If a MTA is configured in a way where no authentication is necessary, i.e., any device on the Internet is able to submit emails, it is referred to as open relay. The responsibility of a MTA when receiving a message is to process it and determine how it should be routed according the result. Therefore, messages consist of the data to be transferred next to an envelope containing all the necessary information required by the MTA, avoiding the need for parsing the whole message. If the desired destination is not available, the MTA is responsible for queuing the messages and retrying the transmission several times before returning a notification indicating an unsuccessful delivery to the origin if still not possible. MTAs make use of three different buffers for processing messages:

- *forward-path*: This buffer is used for storing a list containing one or multiple recipients per message.
- *reverse-path*: The source address of a message is stored here.
- *mail-data*: The mail-data storage buffer is used for saving the mail content data itself.

In order to provide some kind of traceability, MTAs add tracing information to mail message headers as they pass trough the mailing system, like a receiving date or authentication results. When the MTA processing is completed successfully and the recipient is hosted locally, the message is handed over to the Mail Delivery agent 2.1.3. Otherwise, if the recipient is hosted on a remote server, the email has to be relayed to another MTA. Therefore, MTAs also have to implement client functionality next to server functionality. While for other components multiple different protocols are in use nowadays, for MTAs SMTP as exclusive protocol has emerged over time in Internet email context 2.2.

2.1.2. Message Submission Agent

A Message Submission Agent (MSA) [15] is a special version of a MTA. While the primary task of a MTA is to care about the mail transport, MSAs are responsible to provide a message submission destination for User Agents. Usually MTAs are not altering messages except some tracing header fields.

MSAs on the other hand are supposed to correct possible faults introduced by User Agents regarding the standardized message formats. An additional main difference in contrast to MTAs is the responsibility for performing an end-user authentication - a task that became very important with time regarding matters of security and spam.

2.1.3. Mail Delivery Agent

A Mail Delivery Agent (MDA) is often implemented as part of a MTA. The task to be fulfilled by a MDA is the transmission of mails handed over by a MTA into the user's local mailbox, respectively storing the mail until a user accepts it. Additionally, it has possibilities for filtering and manipulating messages before delivery. Similar to MTAs it is in the responsibility for MDAs to attach tracing information to mail messages.

2.1.4. User Agent

A User Agent (UA) is the missing link between users and the message handling system. In order to provide a comfortable user experience they often implement graphical user interfaces. In recent years also web applications got a very important role for UA implementations as alternative to dedicated applications. UAs provide several possibilities for mail handling, like composing new emails, viewing emails, deleting emails or replying to an email. Usually they also provide a local message storage omitting the need for fetching emails repeatedly. If a new message is composed and should be sent to a remote destination, the mail is forwarded by the UA to a MSA via the SMTP protocol. Receiving incoming emails from the remote message storage is achieved by applying the POP3 2.3, IMAP 2.4 or other (less important, often closed third-party) protocols on the other hand.

2.1.5. Mailing Architecture Overview

Figure 2.1 shows all the components linked together while an email is sent from *Sender* to *Receiver*. The *Sender* makes use of a user agent software to compose an email including the provision of a receiving address. When the *Sender* is satisfied with the composed email the UA can be instructed to send the mail to a message submission agent via SMTP protocol after authentication. After reception, the MSA routes the email to the next destination using DNS and MX records 2.1.6 - in the use case depicted within the figure to an intermediate MTA. The intermediate MTA performs routing as well and forwards the mail to the destination MTA. Therefore, the MSA and the MTA each use SMTP for relying the message while making use of their storage capability if necessary, for example to buffer and delay the transmission of emails to a later time if a destination host is not responding. If the MDA is not on the same host respectively integrated within the MTA (otherwise this can be handled internally), the destination MTA hands the message over to the message delivery agent, for example via LMTP (Local Mail Transfer Protocol) - a variation of the SMTP protocol in first instance considering the lack of temporary mail storage capabilities of MDAs. Therefore, a MDA using LMTP denies the reception temporarily if it is not able to handle the message in time. The MDA is finally responsible to deliver the message into the user mailbox from where it can be accessed by the *Receiver*. The access, respectively reception of emails by the *Receiver* can be achieved by employing a user agent software to communicate with the corresponding POP3 or IMAP server picking the emails from the user's mailbox.

2.1.6. Routing

In order to resolve email destination addresses, MTAs have to translate them into IP addresses [16]. For this purpose the MTA makes use of the Domain Name System (DNS). DNS lookups for mailing services are slightly different from regular DNS lookups. Resource records within the DNS responsible for email services are referred to as mail exchanger (MX) resource records. If the DNS server is able to find a MX record according the mail address provided with the lookup request, it will respond either a canonical name (CNAME)

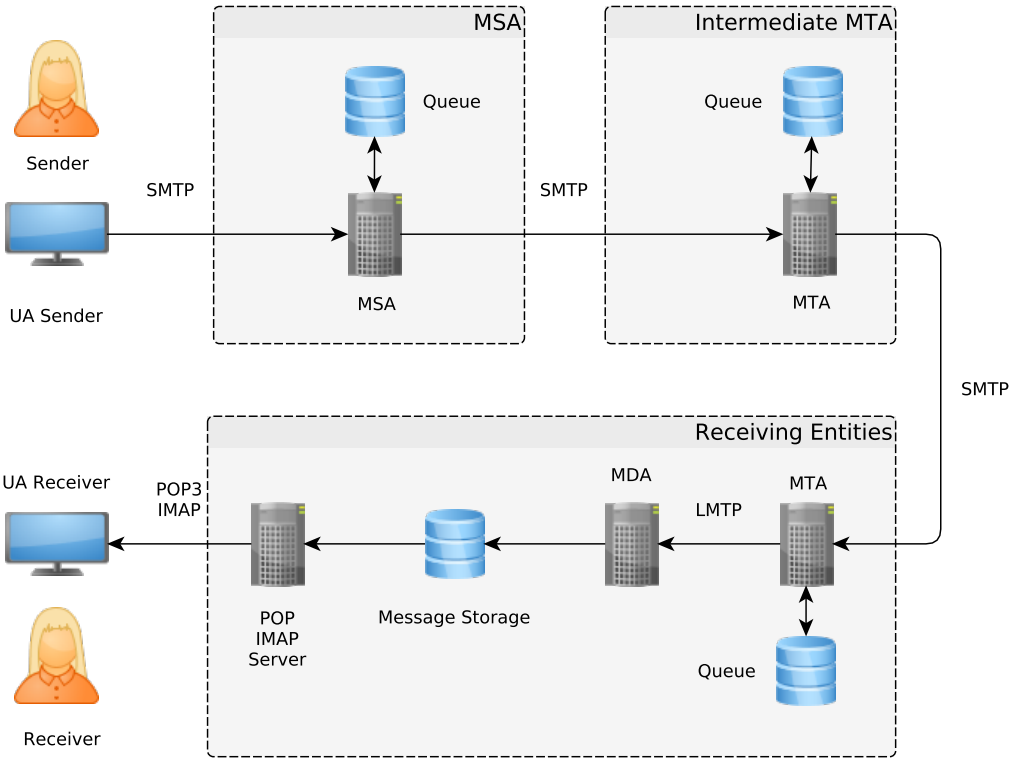

Figure 2.1.: Mail architecture: sending an email.

or one or multiple MX records. Otherwise, if there is no corresponding entry or no DNS server is available, error handling has to be performed; in first instance the domain name of the recipient itself might be used trying to establish a connection or the transfer has to be terminated otherwise, leading to an error message. In case of a canonical name a follow-up DNS query has to be performed providing this CNAME. This might have to be repeated until a response not related to a CNAME is returned. In the end, possibly after resolving CNAME entries, the MTA will be in possession of MX records. While the payload of those MX records contain multiple pieces of information, two values are of particular interest: the mail exchanger and the preference value. The mail exchanger field references to a fully qualified domain name of a server capable of processing emails. The second field on the other hand, the preference value, is used for determining priorities.

This field is necessary as it is possible that there are multiple mail servers available for a domain, usually for high-availability or backup reasons. The RFC defines to handle lower preference values with higher priority; therefore this field is also referred to as distance. In order to continue the MTA has to select one MX record entry for trying to establish a SMTP connection 2.2. For this purpose undesired entries are removed from the list of MX records: for instance the sending MTA itself and records with higher preference values than occurrences of the MTA itself. The remaining MX record entries are sorted by their preference value from low to high. In a consequence the MTA iterates trough this list trying to open a SMTP connection to the target server until successful. In order to establish a connection there is still an IP address of the receiving entity necessary hat the MTA can receive via regular DNS request providing the fully qualified domain name gained from the MX record.

2.2. SMTP

SMTP, defined in RFC 5321 [16] as most recent version, is the protocol responsible for exchanging emails between multiple networks. As basic prerequisite for sending emails an ordered data stream is necessary, usually TCP on port 25, 465 or 587, while other protocols are possible. A SMTP client is responsible to initiate a transaction to a SMTP server via 2-way channel whereby the server address can be resolved via DNS (Domain Name System) and the server can either be the final target referred to as post office (inspired by the analogous postal service), or a relaying station, acting as client itself. Today there are two different kinds of SMTP servers in the field: MTAs (Mail Transfer Agents) for relaying mails and MSA (Mail Submission Agents) for accepting mails from clients only if a user is authenticated. In order to initiate a mail transaction a client has to perform a handshake with the server in first instance. The server responds to all commands sent by the client either with messages indicating a command was accepted, further commands are expected or an error occurred. After the handshake was performed successfully, the mail transaction can be initiated including the delivery of several informations like mail headers, mail content and other metadata using the appropriate commands. After the submission is

complete, a client can either initiate another transaction or shut it down. Data transferred by SMTP is referenced as a mail object, consisting of an envelope and the content. The envelope includes an origin address, one or multiple recipient addresses and optionally appropriate extension data. The content on the other hand consists of header fields (section 2.5) and a body containing the user data to be sent, usually in MIME (Multipurpose Internet Mail Extensions) format (section 2.5.2).

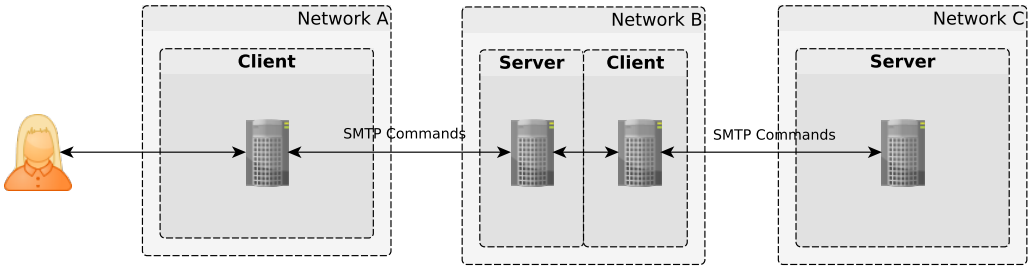

Figure 2.2.: SMTP relaying.

2.2.1. SMTP Procedure

Although there are several commands specified within the SMTP specification, only few of them are used frequently for transferring mails; most of the others are intended for debugging purposes. Every line transmitted, either from client or server, has to be terminated by a `CR` followed by a `LF` character and must not be longer than 512 characters. A typical scenario for transferring a mail from client to server is as follows:

1. **Session Initiation:** A SMTP client application opens a connection to a SMTP server. The server usually provides a welcome message when successful.
2. **Client Initiation:** After receiving a welcome message the client responds with an `EHLO` command, indicating the client's identity and its ability to handle SMTP together with its service extensions.
3. **Mail Transaction:** The mail transaction is initiated by the client when sending a `MAIL FROM` command providing the source mailbox identifier as parameter. The server resets its state and responds with `OK`.

Afterwards the client has to send the `RCPT TO` command providing the recipient's mailbox name and domain as parameter. This command might be repeated several times, depending on the number of recipients while the server each time responds with `OK`. When all recipients are provided, the transmission of the mail data can be initiated by sending a `DATA` command. In a consequence the server expects all succeeding lines as data to be sent until an *end of mail* indicator is appended (a line only containing a dot). When the server receives this indicator, the processing of the received data is performed.

4. **Closing the session:** If the client committed all data either a new mail transaction can be started or a `QUIT` command might be sent in order to quit the session.

Every command sent from client to server is responded by a well defined three digit reply code number indicating the server's state next to some short text messages depending on the implementation. There are several codes available; `250` means everything is ok and the command was accepted whereas `550` indicates that the specified mailbox was not found for example.

2.2.2. SMTP Service Extension

As SMTP was designed already several decades ago the demand for extensions or updates arose over time. In order to maintain backwards compatibility an extension mechanism was established instead of creating a new SMTP version. A client is able to determine if a server provides extensions by sending an `EHLO` command, a variation of `HELO`. The server will either respond a list of extensions it is supporting or an error message if it cannot handle `EHLO`. An interesting standardized extension in context of certified emails might be the *Delivery Status Notification* [17]. It provides delivery status informations like success, delay or failure messages designed to be processed programmatically. In order to handle new commands a registry for SMTP extensions was established by IANA. The procedure for registering a new extension is described within RFC 5321 [16]. For private or research purposes there is also the possibility to create non-standard

extensions by using `x` as prefix for the command name.

2.3. POP3

As SMTP is only responsible for delivering an email to the post office or relaying it to another SMTP server on the other hand, a different protocol for transporting an email from the post office to an end-user is necessary. Over the years two different protocols emerged. The less sophisticated protocol, referred to as POP3 [18] (Post Office Protocol 3), is covered within this section. The protocol enables a client software application, referred to as UA (User Agent), to take delivery of received emails stored in the user's mailbox. Usually the mail is deleted on the server after downloading immediately, although this behavior is in the responsibility of the client application. Similar to SMTP an ordered data stream is essential for the communication, ordinarily TCP via port 110. As for SMTP, commands consist of printable ASCII chars, lines have to be terminated using `CRLF` and reply codes are returned from the server indicating the result respectively server status next to the responses. Unlike to a SMTP server on the other hand is the possibility of returning multi-line responses. In this case a special set of characters indicate the end of a response (`046.CRLF`).

2.3.1. POP3 Procedure

The communication between client and server takes place in three states whereas the first state, the authorization state, is initiated by establishing the TCP connection after the server sent a short greeting message.

1. **Authorization State:** In this state the client has to authorize against the server. Different mechanisms are available to perform an authorization, for example a `USER` and `PASS` command combination. The mechanism provided by the server is not defined, but at least one has to be provided. The server acquires appropriate resources if successful, enumerates all mails contained in the mailbox from one to n, calculates

the size of each mail and returns a positive indicator leading to the transaction state.

2. **Transaction State:** When the session is in transaction state, the client may issue following commands:
   - `STAT`: When receiving this command, the server responds with stats regarding the mailbox. At least the number of messages and the storage space has to be returned while additional information is possible.
   - `LIST`: Using this command a client can request details of a distinct message by providing the message number as parameter or of all messages if no parameter is provided otherwise. The answer contains at least the message number and the according storage space and may contain additional information.
   - `RETR`: The server returns the email corresponding to the message number provided as (obligatory) attribute.
   - `DELE`: The email corresponding to the message number provided via mandatory attribute is marked as deleted, whereas the concrete deletion happens within the update state.
   - `NOOP`: No action, only a positive response is returned.
   - `RSET`: Messages marked as deleted become unmarked.
   - `QUIT`: Causes the initiation of the next state, the update state.
3. **Update State:** In this status the server deletes all messages marked as deleted, releases the resources and quits the tcp connection.

The commands above are the commands a POP3 server has to support at least while it is suggested to implement some additional commands in order to improve the mail handling capabilities of the client: `TOP` returning only the headers of the message instead of the whole content, `UIDL` returning a unique, persistent identification number and `APOP` for an improved, more secure authentication.

2.3.2. POP3 Extension Mechanism

POP3 has not considered possibilities for adding new features in the core specification. Nevertheless an extension mechanism similar to SMTP was

specified for POP3 in RFC2449 [19] providing additional commands including the support for more sophisticated authorization mechanisms and extended response codes. Similar to SMTP it is also possible to add commands using x as prefix for experimental or private purposes.

2.4. IMAP

As mentioned in the preceding POP3 section, two alternative protocols have established over time for transporting mails from the post office to the client user application. Where POP3 stands out regarding its simplicity and efficiency, the protocol characterized in this section, IMAP (Internet Message Access Protocol) [20], is attractive for all the features it provides making the user experience more comfortable. IMAP is able to handle multiple mailboxes per user in a way equivalent to local folders and the intention is to keep the emails on the server, in contrast to POP3, where messages should be deleted immediately after reception, acting more or less as some kind of buffer memory. Therefore, using IMAP only a copy is transmitted from the server to the client when a user retrieves an email, implying the availability of backups if a client device is broken. Additionally, IMAP was designed to handle multiple clients accessing the same mailbox gracefully; a feature gained high importance over the last years as it is common practice to use multiple clients (smartphones, notebooks, etc.) in parallel nowadays. Furthermore, IMAP has integrated support for MIME structures 2.5.2 and the internet message format obviating sophisticated parsing by clients. The protocol offers several possibilities for accessing and manipulating emails on the server. It provides functionalities like searching in mails, creating, deleting and moving mailboxes or checking for new messages by informing the client from the server via notification instead of client polling for example.

2.4.1. IMAP Procedure

In the same vein as POP3 and SMTP, IMAP makes use of text based commands terminated by `CRLF` for the communication between client and server

sent over a TCP stream, usually by occupying port 143. Additionally, IMAP adds the possibility to send literals containing arbitrary binary data. In order to increase flexibility (multiple users simultaneously per mailbox) IMAP differs in the behavior of POP3. Where POP3 always responds to a command sent by the client immediately, an IMAP server might also send responses independently from the commands. For this purpose two protocols are in place: one for the usual client commands and the corresponding response codes similar to SMPT or POP3 next to another protocol responsible for sending data from server to client. Using this approach it is possible to notify multiple clients if a new email was received for example, without requiring a client to initiate some kind of query command. IMAP can handle several commands asynchronously instead of pipelining them like POP3, therefore every command needs a preceding unique random tag enabling the client to assign a response from the server (including the corresponding tag) to a particular command. This results in major performance improvements when performing very computationally commands like searching through mail messages content matching a specific string for example. As IMAP sessions tend to have a high longevity compared to POP3, there is no session timeout defined; sessions are only terminated automatically after inactivity (receiving no command) of at least 30 minutes.

After a TCP connection was established by the client followed by a greeting message from the server, a session is initially in the *Not Authenticated State* (if not already preauthenticated). As IMAP provides a large list of commands and a complete description would overwhelm this document only a subset of the most important commands is described within this section. Moreover, required respectively optional arguments for the commands are neglected here; if required an extensive description can be found in RFC 3501 [20].

1. **Not Authenticated State:** This state roughly corresponds to the POP3 *Authorization State*. In order to authenticate against the server a client can either send a `LOGIN` command providing username and password, or use the `AUTHENTICATE` command offering different and more secure mechanisms for authentication while maintaining the possibility to add new mechanisms. If the authentication is performed successfully, the session state gets forwarded to the *Authenticated State* where a larger set of commands is available.

2. **Authenticated State:** In this state it is possible for clients to manage mailboxes. A new mailbox can be created using the `CREATE` command, they can be deleted (`DELETE`), renamed (`RENAME`), subscribed (`SUBSCRIBE`) respectively unsubscribed (`UNSUBSCRIBE`), listed (`LIST`) next to some other possibilities. In order to act on mailboxes itself the client can select a particular mailbox using the `SELECT` or `EXAMINE` command for read-only access. Those two commands induce a state transition into the *Selected State*.
3. **Selected State:** Acting on particular mailboxes is possible in the *Selected State*. For example, it is possible to search (`SEARCH`), fetch (`FETCH`) or copy (`COPY`) particular messages contained in the selected mailbox, or to set flags indicating a status for a message (`STORE`). In order to switch back from *Selected State* to *Authenticated State* the `CLOSE` command has to be executed.
4. **Logout State:** The *Logout State* can be reached from every other state by executing the `LOGOUT` command. It is also in place if the server gets shut down or the connection to the client is lost. This state is used by the server to terminate the connection gracefully.

The server responds with one out of three possible response types: status responses, server data or command continuation requests. Status responses are predefined strings (*ok*, *no*, *bad*, *preauth* and *bye*) representing the server state after command execution, similar to POP3. Optionally response codes providing additional information can be appended. They may contain a tag initially transmitted by the client request if the response is related to this command making it possible for the client to assign the response or a * character as tag replacement for information not associated with a command. Server data responds return larger amounts of data managed by the server like messages, search results or mailbox lists while those results are always untagged. If a command is not completed yet and additional client data is necessary for processing, the server replies with a command continuation request response on the other hand. This response usually occurs during authentication or responses containing literals.
Unlike POP3 and SMTP IMAP supports several types of data transferred between client and server on protocol level, 13 to be precise. This results in much more complexity regarding the implementation of the protocol itself compared to POP3 while making the development of a client application

more convenient. One of the most significant data types improving IMAP might be literals. They allow sending arbitrary data with few restrictions, especially regarding the length of data being limited to single lines when using other data types. Moreover, message numbers, unique message numbers, message flags (assigning status information to messages (*answered*, *deleted*, *draft*, *flagged*, *recent*, *seen*, ...)), envelope and body structures (providing message meta information) are supported on protocol level next to some minor important data types.

2.4.2. IMAP Extensions

Alike POP3 and SMTP IMAP also provides an extension mechanism while here the `CAPA` command necessary for this mechanism is already defined in the core protocol itself in contrast to POP3 and SMTP. Next to several extensions already defined, new IMAP capabilities can be registered at the IANA IMAP registry by publishing a standard track or RFC. Alternatively non-standard capabilities can be added when *X* is used as prefix for the name. Non-standard extensions without *X* as prefix must not be provided by IMAP servers.

2.5. Email Formats

When sending certified emails, most certainly metadata has to be generated in addition to the encrypted message content of the original message. In the purpose of analyzing the integration possibilities for this kind of data, the standardized format of mail messages is investigated within this section.

2.5.1. Internet Message Format

In order to normalize, respectively standardize messages sent between hosts and different networks, the Internet Message Format (IMF) is defined in RFC5322 [5]. Local storage of emails on the other hand is not bound to this definition. Basically emails consist of simple text lines terminated by

CRLF while a length limit of 1000 characters is required and 80 characters is suggested, each including the terminating CRLF. Although it is not important for the IMF itself to limit the length, the restriction is defined since many implementations of email infrastructure components have problems to handle lines exceeding this limit. The 80 character limit on the other hand is suggested to fit to many graphical user interfaces truncating or wrapping the lines otherwise. RFC5322 [5] defines only 7-bit ASCII characters to be used within mail messages. In order to transfer more complex data like images or other binary data the IMF is extended by the MIME Messages specification 2.5.2. IMF defines two sections from which an email is composed of: the header section followed by the body section separated by a blank line. The header includes metadata like sender, recipient, date and subject in a defined format making it easy for MTAs to parse this information. The body on the other hand contains the payload, i.e., arbitrary data a sender intends to transfer to the recipient. It has to be taken care, that the IMF header and body is not the same header and body used by SMTP: the SMTP header and body might be interpreted as a message envelope while the IMF would depict the message itself.

Header IMF header fields consist of a name field and a body field separated by a colon and terminated by CRLF. In order to handle the 1000 character length restriction there is a mechanism, referred to as folding, available to split the body field into multiple lines. This can be achieved by inserting CRLF immediately followed by a white space character. Folded header fields have to be unfolded before processing by removing the CRLF characters. Using this approach no length limit restrictions are in place for the unfolded fields. It must be considered, that header fields don't appear in a particular order; the order might even change during transmission. While the only required header fields are the origination date field and the originator address field, several other defined fields are possible. A short description of the most important fields:

- *orig-date*: The time when a user pushes the send button within the user agent software - this is actually not the time when the message is transported.
- *from*: The address referring to the mailbox of the message creator.

- *sender*: Address referring to the person sending the message; might be a secretary for instance.
- *reply-to*: Mailbox address indicating a mailbox where replies should be sent to.
- *to*: The receiving mailbox.
- *cc*: Additional receiving mailbox, while this field indicates that the recipient is not the main receiving entity.
- *bcc*: Additional receiving mailbox, while addresses contained in this field are not visible to other recipients in contrast to the cc field.
- message-id: A unique id identifying a message; should be changed as soon as the message content changes.
- *in-reply-to*: The message identifier of a message being replied to.
- *references*: The message content of a message being replied to.
- *subject*: The subject of a message defined by the message originator.
- Trace fields: Fields containing information like received timestamps to trace a message; set by MTAs and MDAs.

Additionally, it is possible to add optional unspecified header fields. The only restriction is that the name differs from fields specified by the rfc. Those fields are uninterpreted per default.

Body The body field is rather simple compared to the well defined and structured header field. Without the MIME extension 2.5.2 arbitrary 7-bit ASCII text may be inserted here without any further restrictions (except the length restriction and CRLF line termination).

2.5.2. MIME Messages

The standard IMF defined in RFC5322 [5] restricts emails to consist of 7-bit ASCII characters for each, header and body of the message. Using 7-bit ASCII leads to problems if other languages than English should be supported as it is not possible to add language specific characters. It is difficult to send non-text data like images, pdf files or other binary data. Difficult means that it is necessary to encode the data in a format conforming 7-bit ASCII (for example hexadecimal) while the receiving entity is still not able

to identify the type of the data. Additionally, it is desired to send multiple pieces of different data in one message - a task not easy to be fulfilled with an unstructured, non-standardized text chunk. Therefore, the IMF was extended by a series of RFC specifications covering Multipurpose Internet Mail Extensions (MIME) to face this gap. The first specification, RFC2045 [21] defines the headers to extend IMF by MIME messages. RFC2046 [22] specifies the structure of the MIME media typing system next to an initial set of media types. The 7-bit ASCII limitation of IMF headers was targeted by RFC2047 [23]. RFC4288 [24] and RFC4289 [25] specify registration procedures for introducing new MIME types while RFC2049 [26] determines conformance criteria for user agent implementations to be MIME conform next to some example MIME messages.

MIME headers The MIME headers extend the mail headers defined by the IMF [21]. While they are generated by the sending user agent, they can be utilized to process the message from the receiving user agent. The following headers are defined:

- *MIME-Version*: This is the only required field to be provided in order to generate a MIME message. It is intended to provide backwards compatibility if new MIME versions are introduced. The most recent version is still 1.0.
- *Content-Type*: This field defines the content type of the data encapsulated within this MIME object. It contains a media type specifying the data format, a subtype for a more detailed specification level and optional parameters useful for processing, whereas the order of the parameters is not significant. New, experimental content types may be added by using a X-token as prefix for the encoding.
- *Content-Transfer-Encoding*: As data fields might contain content not supported by IMF, they have to be encoded for the transfer. The type of encoding is specified within this header field, whereas possible types are 7bit, 8bit, binary and base64. Similar to the Content-Type header field, new experimental transfer encodings can be specified by inserting a X-token as prefix for the encoding name.
- *Content-ID*: A unique MIME entity identifier comparable to IMFs message id.

- *Content-Description*: A textual description of the content that might be useful for the recipient to interpret the data.
- *Content-Disposition*: Determines content information for the user interface. It can be defined if the content should be displayed inline or as attachment for example.
- *Content-MD5*: A MD5 digest providing the possibility for an integrity check of the message content.
- *Content-Language*: A language tag enabling the user agent to programmatically detect the message language.

RFC2047 [23] specifies an extension for the MIME headers to allow non-ASCII characters.

MIME Media Types A number of different data types for MIME messages are defined in RFC2046 [22] including several subtypes. The type contained by a MIME entity has to be defined in the *Content-Type* header field. The majority of those data types like image, audio, video, etc. are not important in certified email context. Therefore, only the types potentially useful for a certified email system are described:

- *text*: This type indicates a MIME entity containing text while different variants are defined by the subtype. The subtype *plain* defines simple text without further interpretation. Additionally, also enriched text and html is possible for example.
- *application*: The *application* top level media type is used for data not fitting into another category. Special applications or user agents supporting this format are necessary to handle this kind of data. Subtypes usually refer to the application name.
- *multipart*: *multipart* is a fundamental property of MIME messages. This type allows the encapsulation of multiple different MIME entities into another entity. Therefore, a body of a multipart message can contain one or multiple parts separated by a delimiter, whereas this delimiter is specified as header attribute. Each part consists of a header similar to a MIME header and a body, separated by a blank line. The header is similar to the MIME message header itself while it is not necessary to redefine the MIME-Version within a multipart field. In a consequence this media type enables the usage of different (or similar,

but separated) data types within a single message. Subtypes can indicate a mixed set of data type parts within a single message (*mixed*), the same data in different data formats (*alternative*), data intended to be displayed in parallel (*parallel*) or data of type message/rfc822 for all subparts (*digest*). The media type multipart may also be used in another multipart body part and multipart messages with a single body part are explicitly allowed.

MIME Extensions For implementing a certified mail system it might be useful to extend the MIME format using the provided possibilities in order to integrate new features in a standard conforming way. The following entities are considered to be extended:

- Content Types: There is no designated registration possibility for new top level types. This can only be achieved by defining a new RFC. In order to extend MIME by additional media subtypes, the according extension mechanism is addressed by RFC4288 [24]. In particular, new subtypes have to be registered at the Internet Assigned Numbers Authority (IANA) providing a central registry. This registration mechanism is necessary to ensure interoperability between different systems. In order to initiate a registration, a proposal has to be composed. Different types of registration can be performed afterwards if accepted:
  - standard: for subtypes of general interest intended to be implemented by the Internet community.
  - vendor: for commercial applications of third party vendors. Subtypes corresponding to this registration type have the prefix *vnd.* in their name.
  - personal: experimental subtypes not to be distributed commercially. Names of this registration type are denoted with *prs.* as prefix.
  - special: unregistered, experimental. Only useful if parties exchanging this type agree on this format. Equivalent to names starting with *X-*, while *x.* is used as prefix for the subtype name in this case.

As alternative, for experimental purposes, arbitrary content types can simply be added by adding *X-* as type name prefix in order to be distinguishable from well-defined formats as already stated within the MIME header section. It has to be taken care that experimental/private types prefixed by *X-* might get lost when sent over gateways filtering those types.

- Transfer encoding: New experimental, respectively private transfer encodings can be added by inserting *X-* as prefix to the new type name of the transfer encoding header field preventing the mixture with new formats in the future. Types prefixed with *X-* might be ignored by gateways depending on their implementation respectively configuration. If standardized encodings should be added on the other hand, they have to be specified as new RFC while the procedure is described in RFC4289 [25].
- Header fields: Additional MIME header fields might only be declared in future RFCs. Alternative extensions are not designated.

2.6. Security

When sending an ordinary email, usually at least three requirements are desired:

1. Authenticity:
   a) A user agent sending an email hast to authenticate against the sending server entity in order to prevent impersonation and spam messages.
   b) A receiving user agent has to authenticate against the receiving server entity in order to assign the correct mailbox, respectively prevent attackers from accessing foreign mailboxes.
   c) Users might be interested in having a possibility to verify server identities.
   d) A recipient might want to verify the authenticity of the message author.
2. Integrity: it is expected that message content is not altered during transmission.

3. Confidentiality: messages transferred over the network should not be accessible for attackers or relaying stations.

The following subsections describe the mechanisms provided by mail protocols targeting those properties. The description is on a rather high level not providing protocol details. For details additional literature is necessary. Also security mechanism in DNS context are not elaborated here, as they are out of scope in certified email context.

2.6.1. Simple Authentication and Security Layer

Originally SMTP had no possibility for client authentication leading to open relays which in turn led to a huge amount of spam messages. POP3 and IMAP had only limited, weak authentication mechanisms like providing username and password in plaintext. Client authentication for SMTP along more sophisticated mechanisms for POP3 and IMAP authentication were specified using their extension mechanisms. Nowadays usually all three major email protocols support the Simple Authentication and Security Layer (SASL) [27]. SASL is a framework intended to separate mailing (and other) protocols from authentication. Several authentication and security mechanisms are provided by SASL and a registry was established by the IANA making it possible to add new mechanisms. Currently approximately 40 mechanisms[1] are specified. Depending on the selected mechanism, SASL can also be used for server authentication, data integrity checking and encryption. Summarized, server authentication (1c), client authentication against SMTP (1a) and client authentication against IMAP and POP3 servers (1b) are fulfilled if appropriate mechanisms are chosen. Integrity (2) and confidentiality (3) seem to be fulfilled when an appropriate mechanism is chosen. Nevertheless, the latter two properties are problematic, see 2.6.3.

2.6.2. STARTTLS

In order to provide integrity and confidentiality for emails when submitted to a SMTP server, respectively fetched from a IMAP or POP3 server, START-

[1] https://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xhtml

TLS can be used to encrypt the data shared between a user agent and the server entities. Relaying SMTP servers can also make use of STARTTLS to protect the connection. STARTTLS basically makes use of hybrid encryption: asymmetric cryptography for key exchange and symmetric cryptography for efficient data encryption. A server using STARTTLS has to provide a certified public key used by the client to encrypt the data, respectively the symmetric key used for encrypting the bulk data. This certificate is also utilized by the client to authenticate the server (1c). For SMTP the integration of this feature was specified as SMTP service extension in RFC 3207 [28]. The IMAP and POP3 extensions on the other hand are treated in RFC 2595 [29]. By using STARTTLS, an encrypted connection on transport level is established. Therefore, all data between client and server is protected regarding confidentiality (3) and integrity (2). Anyhow, confidentiality and integrity properties are confronted with the same problems as SASL 2.6.3. Additionally, in practical use, it has to be considered that most MTAs are not performing server authentication as mail processing has a higher priority than trust.

2.6.3. Security Issues

Both, SASL and STARTTLS, are able to provide encryption mechanisms for ensuring confidentiality and integrity. However, the encryption provided by both opportunities is taking place on transport level. If multiple hops are necessary to deliver an email, the data is decrypted and encrypted again at every single hop. This enables relaying, respectively intermediate entities to eavesdrop and even alter messages. Additionally, it is possible that a message is not encrypted over the full distance; there is no guarantee that every intermediate entity applies an encryption mechanism. This leads to the necessity to provide high trust in all servers relaying an email; a level of trust that usually can not be raised. Both opportunities also lack in the requirement for providing an authentication mechanism of the message originator.

2.6.4. End-to-end Protection

Additional mechanisms or protocols are required for addressing the problems described in the previous section 2.6.3. Therefore, with PGP/MIME 2.6.4 and S/MIME 2.6.4 two alternative technologies have established over time to provide end-to-end protection of messages. For the integration of the necessary data overhead resulting of those security features, new MIME subtypes were defined 2.6.4. End-to-end protection implies that messages are encrypted within the sender's domain, respectively decrypted in the recipient's domain. No intermediate stations are able to get access to the plain-text and alteration would be detected by the recipient. Additionally, a recipient is able to verify the real originator of the message.

MIME Security In order to provide a framework for encryption and signature of messages, two MIME multipart subtypes were introduced by RFC 1847 [30]: multipart/signed and multipart/encrypted, each consisting of two body parts.

- For the *signed* MIME subtype the first body part is an arbitrary message of an arbitrary MIME type defined in its header as usual. The second part consists of the signature over the MIME headers and the message content next to META information for verification. The digest algorithm and the specific protocol in use are provided as MIME type parameter of the surrounding MIME entity.
- Using the *encrypted* MIME subtype, the first body part is used for protocol specific META information necessary for decryption of messages; except the key itself of course. The second body part on the other hand is used to encapsulate the encrypted data itself. Similar to the signature subtype, the specific protocol is defined as parameter of the surrounding MIME entity.

PGP/MIME PGP/MIME, specified in RFC 3156 [31], utilizes the MIME security subtypes defined in the previous paragraph. It provides a hybrid crypto-system allowing to sign and/or encrypt complete messages including possible attachments. To be more specific, PGP/MIME defines three possible subtypes:

- *application/pgp-encrypted*: Encrypted messages can be sent using tis type. A surrounding MIME entity has to be specified as *multipart/encrypted* while a protocol parameter matching *application/pgp-encrypted* is required. The first body part of type *application/pgp-encrypted* contains only one line specifying the PGP/MIME version. The second part on the other hand contains the encrypted message next to some META information and has to be of type *application/octet-stream*.
- *application/pgp-signature*: This type is used to send signed messages. Therefore, a surrounding MIME entity has to be specified as *multipart/signed* while the first body part of this entity consists of the data to be signed next to the second body part of type *application/pgp-signature* containing the signature itself next to some meta signature information. The digest algorithm is defined using the algorithm parameter of the surrounding entity and the protocol parameter has to match *application/pgp-signature*.
- *application/pgp-keys*: Public PGP keys can be transferred using the *application/pgp-keys* type. No multipart entities are intended for this purpose, as the data can be transferred as single block.

In order to sign and encrypt a message, the message can be signed while the result is encapsulated in an encrypted message in a consequence. As alternative PGP/MIME also supports a native combination, resulting in one entity containing the signature and the encrypted text.

PGP/MIME compared to S/MIME S/MIME specified in RFC 5751 [32] is basically an alternative mechanism for PGP/MIME2.6.4 providing the same features in a slightly different way. As the focus of this thesis is not on PGP/MIME or S/MIME and the underlying principle of the end-to-end security mechanism potentially useful for certified emails is already explained by PGP/MIME 2.6.4 a detailed explanation is omitted (for detail information regarding S/MIME, see RFC 5751 [32]). Nevertheless, the major differences on a rather high level might be of interest:

- Although both trust-models are possible with both protocols, PGP/MIME usually makes use of the decentralized Web of Trust while S/MIME focuses on centralized, hierarchical certification authorities for key exchange.

- S/MIME and PGP/MIME use different encoding formats and algorithms.
- S/MIME has a broader acceptance in enterprise context and is accepted by many vendors.
- As S/MIME is already included in most user agents while it is often necessary to install plug-ins for PGP/MIME and the centralized trust model might be more intuitive, S/MIME is probably easier to be handled by end-users.

Security Notes Both, PGP/MIME and S/MIME, lack in one point regarding end-to-end security: Email headers, such as the *from*, *to* or *subject*, are still transferred in plain-text. This circumstance can be mitigated by using MIME security on top of channel security like STARTTLS or similar mechanisms provided by SASL for example. Nevertheless, it has to be taken care that relaying stations or potential man in the middle attackers can get access to this plain meta-data.

3. Certified Mail Basics

Within the security section of the previous chapter three desired properties for ordinary emails were addressed: authenticity, confidentiality and integrity. Those requirements can already be achieved with the ordinary Internet email technology by using a combination of transport layer security (e.g. TLS) and email security (e.g. S/MIME or PGP/MIME). By comparing properties of encrypted email with non-electronic registered mail on the other hand several major gaps can be identified. Therefore, the requirements for a certified email system have to be extended compared to Internet email and it's standardized features. Important properties of analogous certified mails are analyzed in section 3.1 in order to be able to identify gaps of encrypted email in comparison. Those identified gaps are listed in section 3.2, while necessary features and attributes of certified email systems identified by scientific papers are listed in section 3.3. Section 3.4 finally treats Trusted Third Partys and their properties, a trusted parent instance necessary to achieve a fair message exchange.

3.1. Analogous Registered Mail Services

In this section registered snail-mail services are analyzed to provide a basis for identifying the gaps of encrypted emails in comparison with analogous registered letters. Registered mail services differ slightly from country to country, respectively provider to provider while the essential resulting security properties are quite similar. Consequently, this analysis is limited to registered mail services provided in Austria.

3.1.1. Registered Letters

The Austrian Postal Service provides registered mail as a service paired with several additional options[1]. Basically a unique registration number is assigned to every registered letter. This number makes it possible to track letters. When sending a registered letter, the sender receives a confirmation signed by the post office proving the initiation of the delivery process. A recipient on the other hand has to provide her signature in order to take delivery of the letter. Additionally, the recipient has to prove the identity by an official identification document. Two options are available for senders to be notified when the transmission is completed: either the sender can simply be notified by the Austrian Postal Service on a predetermined communication channel (email, SMS or the like) or a delivery receipt signed by the recipient can be returned to the sender making it also possible to verify the identity of the recipient on the other hand. A sender can also determine whether only the intended recipient herself or optionally also an authorized representative may pick-up the delivery.

3.1.2. Governmental Registered Letters

Governmental certified letters are specified within the Austrian Service of Documents Act (ZustG)[2]. In contrast to registered mail, those letters may only be sent by governmental institutions or by law courts. Two types of those governmental certified letters are available: RSa and RSb. The difference between RSa and RSb is basically the regulation that RSa letters may only be delivered to the intended recipient or an authorized person while RSb letters are less restricted and therefore may also be delivered to substitute addressees. A delivery receipt attached to the letter containing sender and recipient information, a signature by the recipient and an Austrian Postal Service indenter is returned to the sending governmental institution after delivery.

[1] https://www.post.at/privat_versenden_brief_oesterreich_zusatzleistungen.php

[2] http://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10005522

3.2. Comparison: Registered Letters vs Signed/Encrypted Email

When comparing email secured by SSL/TLS in combination with email signature and encryption to analogous registered letters several gaps can be identified:

- Using ordinary email there is no possibility available to track sent emails (despite there exist insufficient approaches like DSN 4.2.1). In a consequence a sender can neither check the status of the delivery nor the sender would be aware if an email gets lost.
- A sender has no possibility to prove that an email was sent.
- A sender receives no confirmation that an email was successfully delivered.
- A recipient might deny having received an email at all.

This short analysis already highlights several unresolved gaps in the ordinary email technology even tough applying encryption and other email security mechanisms. The next section deals with desired and missing properties in a more structured way based on scientific papers covering this topic.

3.3. Certified Email Properties

Sending certified emails is not a new topic. Several scientific papers have already investigated necessary requirements for certified emails in more detail than the short hands-on gap analysis in the preceding section. Those papers basically affiliate certified email protocols to fair exchange protocols: the message is only handed off to the recipient if the latter provides his signature on exchange. Fair exchange protocols already have a longer research history as those protocols are also important for many other scenarios. Although there is no clear definition of properties for certified emails two scientific papers from Tauber [33] and [34] have investigated required core properties with practical applicability based on requirements emerged for fair exchange protocols.

- Non-Repudiation: Parties (sender and recipient) must not be able to repudiate the involvement in certain events of a transaction. For this purpose digital evidences bound to specific parties by requiring their digital signature on defined transfer data (including a timestamp) are introduced. Without providing those evidences the delivery execution would be stopped. In certified email context those evidences are:
  - Non-repudiation of delivery (NRD): This is a delivery evidence signed by the MDA making it impossible for a MDA to repudiate the delivery in a consequence.
  - Non-repudiation of receipt (NRR): Sometimes even a stronger evidence compared to the NRD evidence is required approving the reception by the recipient herself instead of the MDA. Therefore, the NRR evidence has to be signed by the recipient on exchange of the message.
  - Non-repudiation of submission (NRS): On the sending side on the other hand a NRS evidence signed by the MTA is generated making it impossible to repudiate a submission at a future date.
  - Non-repudiation of origin (NRO): A signature on the message to be transferred provided by the originator of the message serves as a non-repudiation evidence from the author of the message.
- Fairness: A protocol used for certified email exchange must guarantee fair conditions. In more detail this means that no party (sender or recipient) may prevail over another party. Violating this priciple would for example occure if a sender receives a NRR evidence while the recipient is not able to access the message in plain. Furthermore, this means that after a transaction is complete every party has its expected messages respectively evidences - otherwise no one receives the expected items.
- Non-Selective Receipt: When a recipient is notified about a new incoming certified mail she must not be able to repudiate the reception by this point. Therefore, the recipient has to sign the NRR evidence before getting access to the message. Otherwise it would be possible to select messages to be received based on metadata, e.g., the sender address, while rejecting undesired mails, e.g. from the financial department on the other hand.
- Timeliness: In order to provide determinism, a timespan has to be

defined within a certified mail has to be delivered successfully or the transfer is aborted on the other hand.
- Verifiability of TTPs (3.4): If a TTP misbehaves there must be a possibility for the deceived certified email participant to prove this misbehaviour.
- Confidentiality: Although this property is optionally also available for regular email, certified email has to provide confidentiality by encrypting the message content.
- Effectiveness: A transfer of a certified email should be executed as effective as possible with the minimum number of required claims and if a transaction is executed fair as expected it should not be necessary to interact with the TTP.

3.4. Trusted Third Partys

For resolving possible ambiguities, respectively ensuring the required fairness property, many fair exchange protocols make use of a so-called Trusted Third Party (TTP), an independent entity trusted by all system participants. This entity is necessary as systems without parent instance can lead to word against another's situations. Although there exist other solutions for this situation without TTP, at least in laboratory conditions, those solutions have turned out to be impractical either because of unrealistic requirements, non-deterministic results or huge communication overhead [33]. Basically TTP solutions are divided in three categories with different properties making them more or less applicable:

Inline TTP An inline TTP is the simplest solution at first glance. Every step of a transaction between A and B has to be sent through the TTP while letter is processing the message and forwarding it to it's indented destination. Therefore, an inline TTP acts basically as proxy.

Online TTP Similar to an inline TTP, an online TTP is involved in every transaction. However, there is a major difference between both types: online

TTPs are not involved in every step of a transaction whereby the TTP acts as mediator, for example to exchange session keys. The rest of the communication per transaction takes place without TTP interaction directly between the communication parties.

Offline TTP Communication with offline TTPs on the other hand is only necessary if some kind of inconsistency occurred during the transfer of a certified email, respectively if some kind of dispute has to be resolved. Otherwise, as applicable in most situations, it is assumed that all parties behave fair while there is a possibility for resolving conflicts otherwise.

3.4.1. Trusted Third Party Property Comparison

The selection of a TTP type can be made conditional on some properties distinguishing them:

Communication Overhead Employing TTPs requires communication overhead, as connections to an additional service have to be established. Therefore, reducing communication with TTPs leads to better performance of certified mail systems. Obviously offline TTPs are superior to other solutions as communication is only necessary in special cases while inline TTPs demands most resources as communication is necessary multiple times per transaction. Nevertheless, it has to be considered that message sizes and necessary communication steps between clients increases using offline TTPs compared to online and especially inline TTPs.

Resources Inline TTPs have to process entire messages and provide storage for messages and associated data like timestamps and audit information. In a consequence they have comparable high infrastructural requirements. Offline TTPs on the other hand are usually not involved in the communication and need only limited resources for processing messages therefore. Online TTPs are in perspective of efficiency between inline and offline TTPs: While they do not have to process entire messages, as the sending and

receiving party communicate directly, they still have to do some processing per transaction.

Convenience of Integration The concept of inline TTPs is easy to integrate in a certified email system as they could be included as part of already available components like MTAs for example. The integration of online and offline TTPs in the already existing infrastructure is more complex on the other hand as new components have to be added.

Security Inline TTPs are able to decouple the recipient from the sender as they are acting as a proxy and can therefore control the message flow hiding a sender for example when processing the message. On the other hand this is a serious drawback as they would have to be fully trusted being able to discard, read or even alter messages. Online TTPs require less trust as they are not able to modify, read or discard messages (depending on the implementation and configuration). Nevertheless, they are still able to delay, respectively prevent message exchange. Finally, offline TTPs require even less trust than inline TTPs as they are only involved in special cases. Therefore, during a regular transaction they are neither able to read, alter, discard or delay messages.

4. Related Work

When evaluating ambitions for introducing certified e-mails it can be observed that there are already several solutions operating in the field. In section 4.1 those services are investigated. Additionally there are lots of scientific papers available researching new solutions, algorithms and protocols. Those papers are considered closer in section 4.2.

4.1. Certified Mail in the Field

Many certified email systems are already available in the Internet whereas those solutions can basically be separated into three categories:

- Governmental: Governmental certified mail systems usually have to be certified, respectively authorized by a governmental institution and are intended to be used for delivering mails to citizens on a high security level; a task that is otherwise costly and time-consuming.
- Private: Certified mail systems are also shaping up well for the private sector. While those systems don't have the requirement to be authorized by a governmental institution, they still follow a defined set of rules. The intention behind following those rules is that systems of different providers can interact, that users can assume a well-conceived, secure system and that there is a possibility to exchange mails between different providers.
- Third Party: The last category are third party solutions provided by private companies or other independent institutions. Usually they are not following a defined rulebook, the specifications are closed and users are not able to exchange mails with other providers.

The implementations and protocols of available certified email systems, especially for the governmental sector, differ from country to country, each providing own solutions. As it is not possible within this thesis to investigate all those solutions in detail, the focus here is on the certified email systems operating in Austria. More solutions from different countries were examined by Tauber in *Cross-Border Certified Electronic Mailing* [33].

Certified Mail in Austria Currently five companies authorized by the Austrian Office of the Federal Chancellor provide governmental, certified electronic mails in Austria[1] considering the Austrian Service of Documents Act (ZustG)[2], whereas the technical specification requirements for providers are publicly available[3]. Summarized the most important points of those specifications on a high level:

- For registration and authentication, users either have to provide their identity by the Austrian mobile phone signature or the Austrian citizen card.
- The Austrian mobile phone signature or the Austrian citizen card is also used to provide integrity and authenticity by appending advanced electronic signatures to certified mails.
- If a user provides a certificate (optionally), all certified mails have to be encrypted.
- When a certified mail is delivered, the according user is informed about the reception via an ordinary email, a SMS or a postal letter after a designated timespan. At the same time a transmission confirmation is automatically returned to the sender, although this confirmation does not reveal information about the actual reception of the receiving entity.
- Immediately when a user logs in after the notification, a return receipt is generated and returned to the sender.
- A LDAP based registry referred to as Zustellkopf is specified making it possible to deliver certified emails to different providers depending

[1] https://www.bundeskanzleramt.at/elektronische-zustellung

[2] https://www.ris.bka.gv.at/GeltendeFassung/Bundesnormen/10005522/ZustG%2c%20Fassung%20vom%2019.03.2017.pdf

[3] https://www.ref.gv.at/AG-II-ZUSE-Zustellung-Spezifik.2822.0.html

on where the user is registered.
- There is also an optional specification for accessing certified emails by regular email clients.
- The specification is not limited to governmental mail: Delivery initiated by private parties is considered as well.
- A governmental certified mail is automatically classified as delivered after the second notification was sent.

For the private sector, Austria plans to obligate companies providing an electronic mailbox for governmental mail by year 2020[4].

Third Party and Private Certified Mail Service Systems Additionally, private and/or proprietary third party mail systems introduced by several companies and institutions are available. While those services are not necessarily bound to certain countries, the main problems of those systems are the closed, private specifications and/or the missing ability to interact with other certified mail systems. Users have to register at those communication silos while a high trust in the provider is necessary.

Certified Mail Services in the Field Currently the following services for certified mail messages are available in Austria following the specifications provided by the Austrian government - third party and private solutions are not considered as the specifications are not publicly available or the lack in dissemination and/or interoperability classifies them as not significant:

- Bundesrechenzentrum GmbH: Elektronischer Zustelldienst. The Elektronischer Zustelldienst allows only the reception of governmental certified mails. It was approved in 2009 as the first player in the field.
- Österreichische Post AG: meinbrief. This service was introduced by the Austrian Post one year later in 2010. Different from Elektronischer Zustelldienst, this service also allows private delivery.
- Postserver Onlinezustelldienst GmbH: E-Zustellung. Approved in 2012 this is actually the only solution providing the possibility to

[4] https://www.ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2017_I_40/BGBLA_2017_I_40.html

send electronic mails as specified by the WKÖ rulebook [5], a private extension of the the governmental specifications. The service is also certified for receiving governmental mails following the specification by the Austrian government.
- sendhybrid ÖPBD GmbH: eVersand. Another alternative for the reception of governmental certified mails, approved in 2014.
- Hpc DUAL ZustellsystemeGmbH: Briefbutler. This is the most recent service and was approved in 2017. Similar to meinbrief this service additionally allows non-governmental delivery.

Conclusion Proprietary third party certified mail systems can not be investigated as the specifications and implementation details are not publicly available and private solutions result in communication silos. When analyzing the specifications following the Austrian Service of Documents Act on the other hand, the following conclusions can be drawn:

- It can be observed that those systems are not inter-operable with the ordinary Internet email infrastructure, although an optional reception via IMAP is specified by the Austrian government and several other primitives respectively technologies of the Internet email infrastructure are employed, like MIME messages for encrypted certified mails. All participants have to register a new account within the chosen certified mail system; users are not able to use the system with already existing email accounts and it is not possible to exchange certified emails with ordinary email services.
- There is no TTP available allowing to independently monitor or verify the MTA behaviour: on a closer look, the MTAs overtake the role of inline TTPs leading to a high trust requirement for users.

[5] http://www.ezustellung.at/downloads/e-Zustellung_Rulebook_V12_03-2015.pdf

4.2. Certified Mail in Research

Different attempts were made by the research community to provide a certified email solution. One approach is to extend the current Internet email protocols using their designated extension mechanisms. The other approach category suggests an additional protocol layer above the Internet email protocols.

4.2.1. Extensions for the current Internet Email Protocols

Extending current Internet email protocols like SMTP, POP and IMAP to achieve certified emailing seems the most obvious approach. Next to some minor important protocol extensions, the following are the most relevant alternatives: delivery status notifications [17] specify a method to extend SMTP in order to return notifications for determined delivery states allowing to track emails. This extension is limited to report the delivery into the recipient's mailbox; it is not possible to return information if an email is actually picked up or even read by the recipient. At this point another extension joins the game: message disposition notifications. This kind of notification is realized as extension for the POP protocol [35] respectively the IMAP protocol [36], while the intention is to return a notification to the sender as soon as the recipient accesses the email. An alternative approach was undertaken by extending S/MIME by signed receipts [37]. Nevertheless, all those approaches are facing problems achieving the requirements for certified emails. Apart from the fact that those extensions are not mandatory, recipients are able to prevent the return of the notification or receipt, hence are able to repudiate the reception for example. As there is no TTP in place, this situation can never be resolved.

4.2.2. Certified Mail Protocols on top of Internet Email

With time numerous certified email protocols on top of conventional mailing protocols were investigated and published, each claiming to be the best solution. Therefore, this analysis is organized from a bottom-up view excluding

branches not fitting the needs to fulfil the claimed requirements to implement a practical certified mail solution. First of all solutions without TTP are not further considered as those protocols are classified as impracticable [33]. A protocol using an inline TTP was presented by Zhou and Gollman [38]. While this protocol lacks in fairness also another important characteristic appears: inline TTPs are involved in every single step of the protocol. When the Internet email infrastructure is able to handle very high loads based on it's decentralized design, this is difficult to achieve for TTPs leading to bottlenecks. Therefore, the same authors reduced the TTP involvement in a different protocol making use of an online TTP not requiring TTP involvement in every step [39]. Abadi et al. [40] proposed an alternative protocol using a lightweight online TTP without requiring a public key infrastructure. Although performing better compared to solutions employing inline TTPs, still an involvement per transaction is required for online TTP solutions being not very efficient. As this inefficiency can also be expected for similar online and inline TTP protocols, the focus for his thesis is set to offline TTPs from now, also referred to as optimistic approaches. Optimistic certified mail protocols were introduced by Zhou and Gollmann [41]. Based on this idea numerous optimistic protocols were developed. Many of those protocols turned out to have security problems or cannot ensure fairness. Promising solutions were introduced by Wang et al. [42][43][44]. But those protocols and similar approaches were not designed having in mind specific deployment scenarios like mounting them on top of the existing Internet email architecture. Instead, they assume direct communication between sender and recipient. Mapping those protocols to Internet email often seems difficult or impossible as the infrastructure respectively the participating entities (UAs, MTAs, etc.) are already defined. In a consequence the remainder of this section focuses mainly on optimistic certified email protocols potentially fitting the Internet email infrastructure and making use of TTPs for resolution. A optimistic protocol fitting the Internet email architecture very well was introduced by Blundo et al. [45]. It covers several requirements necessary for certified emails but has some shortcomings: the TTP is not verifiable and selective receipt is not prevented. Liu et al. present a different protocol having the apperance to be very practical on the first glimpse [46]. Nevertheless, following their approach it is required for the TTP to contact end users for resolution. This is not practical as end-users would have to register at the TTP and would have to be online when the resolution is

performed. A very promising protocol comes from Draper-Gil et al. [47]. It's design allows to implement an optimistic, practical certified mail system while also fulfilling the necessary certified mail requirements.

5. Technology Evaluation

Scientific papers propose several solutions for enhancing widespread email technology with certified mail functionality 4.2. Nevertheless, only few of them fulfil necessary requirements. The protocol to be implemented is evaluated in section 5.1. Furthermore, the selected protocol should not be based on email software built from the scratch - instead already existing software should be extended proving also the ease of integrating the new functionalities. Software suitable to be extended is evaluated in section 5.2.

5.1. Certified Email Protocol

In first instance the certified mail protocol to be implemented has to satisfy the required attributes defined in section 3.3: Repudiation must not be possible, fairness has to be ensured, selective receipt must not be possible, timeliness must be guaranteed, the TTP must be verifiable, the certified mail must be confidential and the protocol has to be effective. Additionally the protocol should fit the current Internet email infrastructure. To be more detailed, it should match the communication parties coming with this infrastructure: users, user agents and message transfer agents - no direct communication between two end-users is possible. Furthermore, the resulting system should be scalable similar to the email infrastructure. Hence, as already mentioned in section 4.2.2, optimistic protocols using offline TTPs are encouraged. One protocol appearing to fit all those requirements was introduced by Draper-Gil et al. [47].

5.2. Software Evaluation

In order to demonstrate the practicality, the certified email system to be implemented should preferably be integrated in already existing software components. Software components to be used and extended are particularly a user agent, a MSA/MTA and the basic protocols between user and user agent respectively all other participating parties.

5.2.1. Basic Mail Protocols

While for sending emails from the user agent to a MSA or MTA, respectively handing off emails from MTA to MTA, it is clear to use the SMTP protocol (2.2) owning to the lack of alternatives, things look different for receiving emails from the message storage by the receiving UA. For the letter case there are basically two possibilities: using the rather simple POP3 protocol (2.3) or the sophisticated IMAP protocol (2.4). From a point of view on feasibility it is possible to integrate a certified email protocol in both alternatives. Nevertheless, many things have to be taken care when integrating the protocol in IMAP as there might be multiple mailboxes per user, emails remain on the server instead deletion, multiple clients might use one mailbox, mails on the server are affected by multiple protocol commands etcetera. Although those features enhance the user experience they make it rather complex to integrate the certified email protocol as things have to be kept in synchronization leading to major implementation effort while not changing the outcome of showcasing the certified email protocol concept per se. In a consequence this work will focus on the POP3 protocol.

5.2.2. Mail Server Software

Many server-side mail software solutions are available on the market. A survey of the most used mail servers employing MX record lookups as indicator[1] visualizes that the marked is shared mainly between two server

[1] http://www.securityspace.com/s_survey/data/man.201706/mxsurvey.html

implementations nowadays: *Exim* and *Postfix*, while other options have very less and decreasing market-share. Nevertheless, for this prototype implementation other properties than the market-share might be more important as it should focus more on demonstration than providing an implementation optimized for productive usage. Desired requirements for server-side mail software used as base for the certified mail protocol integration are listed below:

- **Programming Language:** Email servers found in productive environments usually make use of low level programming languages like *C* for performance reasons and are optimized for high throughput. The disadvantage of using a low level programming language on the other hand is much more implementation effort compared to high level programming languages as *Java* or *C#* for example. For demonstrating a prototype high level programming languages are more suitable as major effort would be necessary to integrate certified mail protocols in highly optimized low-level (in terms of programming language) environments.
- **Licence:** A good deal of email server software solutions are based on proprietary licenses limiting adaption respectively usage by third parties. Therefore, a license providing access to the source code and allowing all necessary adaption work is a requirement for the selected server software.
- **OS support:** Although not a hard requirement, as Linux is commonly known to be the most used operating system in server context it would be desirable that the software also supports Linux, respectively is able to be executed on multiple platforms instead being tailored to a single platform.
- **Provided Features:** While some email server solutions provide a complete package of email services including SMTP, POP3, IMAP and mailbox for example there are also solutions, often used in productive environments, providing only specialized services like SMTP only in case of *Exim* and *Postfix*. For this certified email protocol prototype a server solution including all necessary components would be desired.

A list comparing the desired requirements of available mail servers was added to the appendix B. The result shows that more than half of the mail

servers is not useable for implementing a certified email prototype due to proprietary licenses leading to closed source code in a consequence. The remaining mail servers mostly use a low level programming language and/or provide only one type of protocol. As most suitable software in context of developing a certified email prototype system basically one option remains: *Apache James*.

5.2.3. Mail Client Software

Similar to server-side mail software applications also many client software applications are available, while web-based products displaced native clients more and more within the last years. As major changes are necessary on the client side to integrate certified email (additional user interfaces, additional database fields, adaptation of the email protocols on client side, etcetera) a new lightweight web-based email client is developed to provide possibilities for compiling and sending certified emails.

5.2.4. Client Mail API

In order to communicate between client and server a library able to handle email protocols (SMTP, POP3, IMAP) is necessary. A major requirement for this component is its extensibility, hence source code access, as at least some minor changes will have to be performed. For this purpose the JavaMail API seems to be a good choice[2].

[2] http://www.oracle.com/technetwork/java/javamail/index.html

6. Certified Email Protocol

The certified email protocol introduced by Draper-Gil et al. [47] promises effectiveness, fairness, timeliness, verifiability of TTP, confidentiality, non-selective receipt, non-repudiation and makes use of an offline TTP. In order to match the Internet email infrastructure it is designed for four participating entities: sending UA and MTA respectively receiving UA and MTA while any number of intermediate MTAs between sending and receiving MTA are possible. Additionally, a TTP is required while for resolution two-way communication is necessary between MTAs and TTP respectively one-way from UA to TTP. This implies that some kind of registration of MTAs at the TTP is necessary. Relying on an offline TTP leads to two sub-protocols: The delivery sub-protocol executed for common certified mail transfer without special incidents and a resolution sub-protocol in case some kind of irregularity has to be resolved. The protocol's security proof requires the following assumptions:

- The TTP is classified as semi-trusted, i.e. it can misbehave but not collude with other parties.
- Users trust the MTA where their email account is registered.
- The communication channels between users, UAs and MTAs are unreliable while they have to be operational between TTP and it's communication partner.
- The sending UA has to know the receiving UAs public encryption key.

6.1. Delivery sub-protocol

The delivery sub-protocol is the main protocol for message exchange and if no error occurs during transmission it is the only one to be executed

promising effectiveness. On a high level the following steps are performed (for more details see [47]):

- The sending UA generates a random symmetric key k, encrypts the message to be transferred using this key ($E_k[C]$) and encrypts the symmetric encryption key using the asymmetric public receiving encryption key ($PK_{UA_B}[k]$). Additionally, a time-limit t_d for the maximal transfer duration is chosen. All those data chunks are transferred together with a NRO evidence to the sending MTA as M_0. The sending UA obtains a NRS evidence from the sending MTA in exchange (M_2).
- The sending MTA forwards all data except the encrypted symmetric key k to the receiving MTA (M_3) while the latter is transferring this data in the receiving users inbox. The symmetric key has to be stored by the sending MTA for later usage.
- The receiving user can fetch the encrypted certified mail from the according mailbox (M_5). In order to get the key for decrypting the message a NRK evidence has to be generated and sent to the receiving MTA (M_6).
- The receiving MTA forwards the NRK evidence to the sending MTA (M_7) and receives in exchange the encrypted symmetric key and the NRO evidence (M_8).
- After receiving the encrypted symmetric key k, the receiving MTA returns a NRD evidence to the sending MTA (M_9).
- Finally the receiving user agent is able to decrypted the message as it is in possession of the private key SK making it possible to decrypt the symmetric key k. Furthermore, the receiving UA obtains a NRO evidence (M_11), and the sending UA a NRK and NRD evidence (M_13).
- M_1, M_4, M_{10} and M_{12} are synchronization requests.

For a better illustration the delivery sub-protocol sequence is depicted in Figure 6.1. If the maximal transfer duration is exceeded, MTAs stops forwarding or processing new messages: the transfer is cancelled. Different to the evidences defined in 3.3 the protocol does not include a NRR evidence, while a NRK (non-repudiation of knowledge) evidence was introduced. The authors assume that the combination of NRK and NRD also satisfies the NRR evidence.

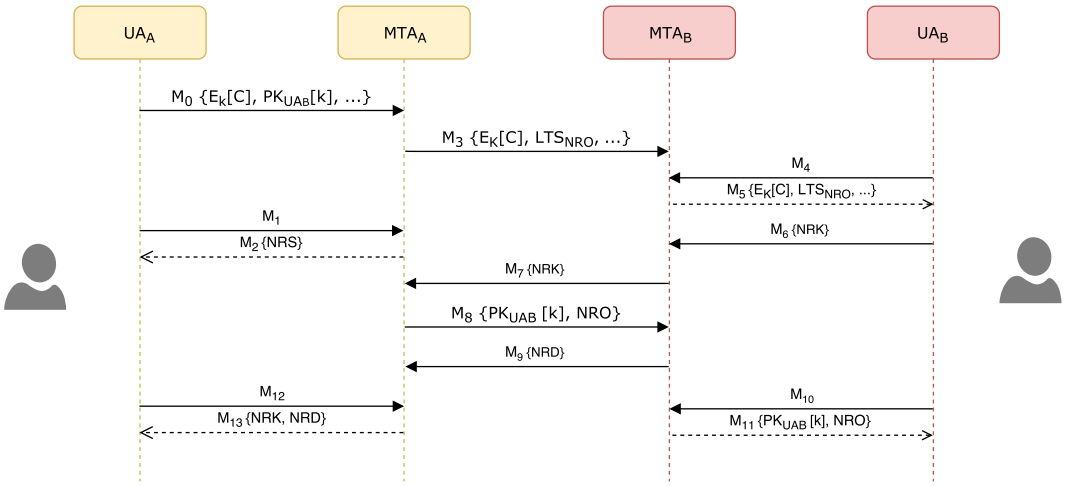

Figure 6.1.: Delivery sub-protocol: **A** sends a message to user **B**. k: symmetric key, C: message, $E_k[C]$: symmetric encrypted message, PK_{UA_B}: public key of user agent B, $PK_{UA_B}[k]$: symmetric key encrypted with public key of user agent B, LTS_{NRO}: NRO timestamp.

6.2. Resolution sub-protocol

The resolution sub-protocol is activated if a to be defined time-limit is exceeded for receiving evidences or other messages are expected but not received. Basically a request is possible for every potentially lost or withhold message from the delivery sub-protocol. UAs and MTAs are able to request resolution by the TTP if they can proof that they are privileged to receive the according resolution data by passing attributes they must already have received. The TTP on the other hand is able to collect information about the transfer stored at the sending and receiving MTA. Based on the data received from the MTAs the TTP might also generate affidavit evidences if enough information is available. If the resolution is successful, the requesting entity is able to pursue the delivery protocol. If the delivery time-limit is exceeded, the TTP answers only with information it already collected - it does not contact MTAs any-more at this point of time. Furthermore, if the TTP is not able to resolve the request, the TTP can cancel the transfer. For detailed information about all possible resolution requests and responses, see [47].

7. Certified Mail Implementation

The implementation chapter points out design decisions, implementation details and how challenges were mastered to integrate the certified email system in already existing basic Internet email components. Therefore, all important entities are analyzed each for itself. An overview of all major components playing a role in the certified email prototype system and their interaction possibilities is shown in Figure 7.1.

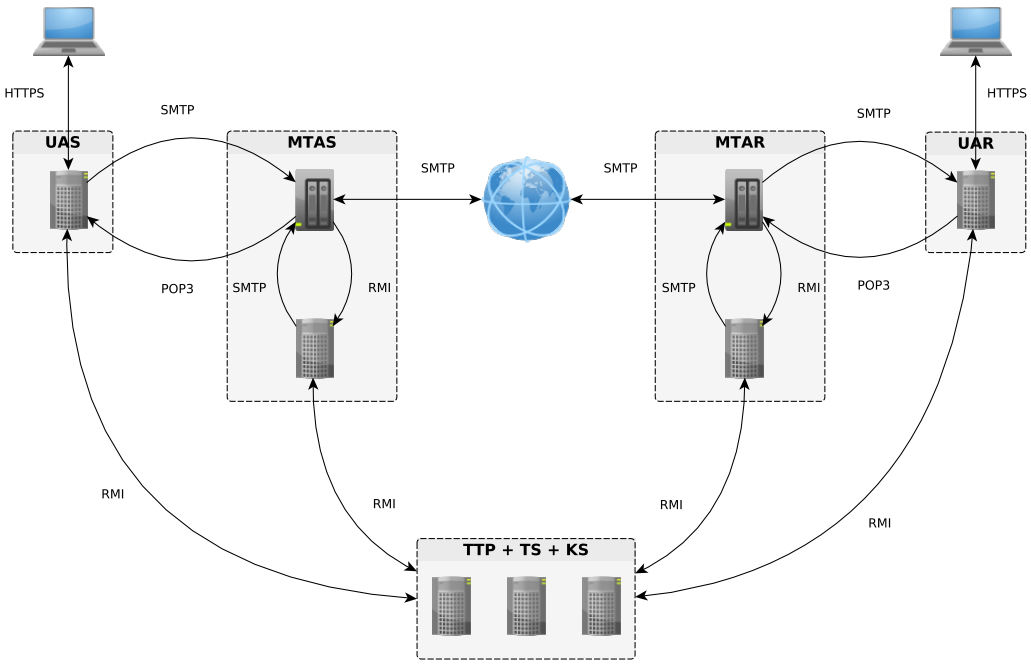

Figure 7.1.: Certified mail system prototype overview.

7.1. Mail server

A core component of a certified email environment is the mail handling software on the server side. The evaluation in section 5 points out that Apache James[1] can best satisfy this project's requirements, i.e., being licensed under a license allowing the modification of source code, being implemented in a higher level programming language like Java, being compatible with Linux and supporting the POP3, IMAP and SMTP protocol (5.2.2).

7.1.1. Specification

Apache James is a modular mail enterprise server implemented in Java. It is licensed under Apache License, Version 2.0[2] providing access to the source code and granting all necessary permissions to modify source code. Components provided by James are amongst others server side end-points for protocols like SMTP, LMTP, POP3 and IMAP, a mail storage implementation together with a mail storage API, administration tools and Mailets: extensible and pluggable email processing agents.

Mailets An important component of Apache James are so-called Mailets[3]. Mailets are intended to provide flexible, independent mail processing functionality covering a wide range of purposes. In order to activate a Mailet it has to be referred and configured in the *mailetcontainer.xml* configuration file. Next to specifying the Mailet class itself also a reference to a so-called Matcher is required. A Matcher is a unit determining if an email conforms to specific requirements defined by this Matcher. If the Matcher takes effect, it initiates the processing by the corresponding Mailet. Several default Matchers and Mailets are already provided and used by James out-of-the-box while it is easy to extend this repertoire.

[1] https://james.apache.org/
[2] https://www.apache.org/licenses/LICENSE-2.0
[3] https://james.apache.org/mailet/index.html

7.1.2. Adaption

It is desirable to implement a certified-mail solution that is portable to other already existing email servers without major effort. Therefore, it is attempted to keep the changes within Apache James itself on a minimum and source the certified mail processing out to an independent, reusable application, a so-called certified mail service 7.2. While the certified mail service application might either be executed on the same server as Apache James, it is also possible to re-locate it to an independent remote server. To achieve this objective, i.e. to undock the certified email processing from James, a Matcher is implemented scanning emails for certified emails. When a certified email is identified by the mime type *multipart/certified*, it is forwarded to a corresponding certified mail Mailet. This Mailet in turn is responsible to establish a connection to the certified mail service, to forward the filtered certified mail and to cancel the email transmission to the next regular destination. The forwarding of filtered certified emails is performed using Java RMI (Remote Method Invocation). In order to provide a less platform dependent protocol (as RMI is bound to the Java platform), this protocol might be replaced in the future with a different, more ubiquitous protocol like JSON over a plain socket for example. Beside the message processing another adaption is necessary to be integrated in Apache James: The implemented protocol requires synchronization messages and responses to be signed. Therefore, as POP3 was chosen as synchronization basic protocol, an adaption is necessary on the server-side POP3 component to verify synchronization requests signed by the recipient and to sign synchronization responses, primarily intended to ensure fairness for the users. It has to be considered that this minor POP3 adaption leads to the fact that the resulting certified mail system is not fully compatible with the originally POP3 protocol anymore. Furthermore, it is necessary to store synchronization request and response timestamps associated to the according certified email. In order to keep the changes in the server software minimal and to keep the responsibilities at the designated places, this signature functionality was outsourced to the certified mail service as well, similar to the delivery sub-protocol mail processing. Summarized, by outsourcing all certified email components, the adaptions to be performed on the mailing server are kept minimal making the effort to integrate it in other mail server software

very low. No additional database schemes or additional dependencies are required.

7.2. Certified Mail Service

The certified mail service is the centerpiece of the delivery sub-protocol and plays also an important role for resolution in case of irregularities. Two-way communication with its corresponding MTA is possible, while the incoming communication is handled via RMI, respectively the outgoing communication via the SMTP protocol. Furthermore, the certified email service is able to contact the TTP for obtaining timestamps, public keys and to request resolution if necessary 7.5. Aside it has to provide resolution information if requested by the TTP. For this prototype implementation all those communication channels were realized using Java RMI. In order to provide a less platform dependent protocol (as RMI is bound to the Java platform), this protocol might be replaced in the future with a different, more ubiquitous alternative protocol. Summarized the certified mail service has four main jobs:

- **Certified mail processing**: A main purpose of the certified email service is to process forwarded certified mail messages originating from the corresponding MTA. Hereby signatures of received messages have to be validated as well as received evidences. If an evidence is invalid, the certified mail service must not continue. The certified mail service itself is not able to initiate a transfer cancellation directly. Nevertheless, either a cancellation is performed as a consequence of the invalid evidence when the resolution process is started by any party caused by not receiving expected data or the transfer times out invalidating the transfer if no resolution process is initiated otherwise. Furthermore, the certified mail service is responsible itself to sign messages before they are sent. Hereby messages intended to be transferred to the users inbox lead to a special case, as they have to be signed at the point in time when returned to the user after the synchronization request instead of the time when they are moved to the MTAs

inbox. In succession those messages should not be signed directly after processing.

- **Sychronization response signatures**: When a user agent performs synchronization using the POP3 protocol, Apache James recognizes by the MIME type if the message is a certified email and contacts the certified email service in a consequence. The certified email service is responsible to sign the response. Furthermore, it is responsible to detect if the synchronization response is the specific message $m5$ (the encrypted certified email): In this case the synchronization request (timestamp and signature) and response timestamp have to be stored in the database if resolution is necessary at a later time.
- **Provide resolution information**: The certified mail service has to provide information about a certified mail transfer in case the TTP contacts the certified mail service for resolution purposes. Therefore, all transfer data necessary for providing resolution information has to be stored in a database for later availability.
- **Initiate Resolution**: The certified email service is executing a scheduled job detecting if awaited responses for a certified mail transfer are overdue according to some configured time values. In this case the certified email service contacts the TTP for resolution by providing the necessary information. Depending on the answer of the TTP the certified mail service can either cancel the transfer or, if resolution is successful, initiate the continuation of the certified mail transfer again.

The certified mail service is responsible to persist all data gained during a certified mail transfer in a database for potential later resolution. Resulting from the protocol design it has never the possibility to get access to the certified mail plaintext. Special attention was also paid to continuation of a certified mail transfer after resolution: Caused by ordinary delays it could happen otherwise, that an entity requests resolution for a specific message. If the resolution was successful, the mail transfer is continued. Nevertheless, the original expected message could arrive at the resolution requesting entity at a later point in time caused by an ordinary delay. Without protection mechanism it could happen that multiple copies of the same certified email are transferred otherwise.

7.3. User Agent

The user agent is the most important component in the end-users perspective as it represents the interface between the certified mailing system and the user. It is responsible to provide a database for storing certified emails and a graphical user interface allowing to send and receive certified emails respectively to provide status information about the transfer. Additionally, a task periodically checking if the resolution protocol should be started is a mandatory part of the user agent. As native applications can be turned off preventing the periodic task execution, web-based always-on applications are desired for a certified email user agent. This also corresponds to the zeitgeist, as user agents are more and more shifted towards web-applications in general. In a consequence it is not purposeful to extend common native applications like Thunderbird or Microsoft Outlook. Instead of integrating certified email functionality in an already existing user agent a new, lightweight, web-based user agent was developed (see Figure 7.2). Compared to integration in already existing software, this approach allows maximum control and access to the application avoiding possible limitations, hence reducing integration effort while increasing testability - properties ideal for a prototype. The web-application implementation is based on the JavaServer Faces framework[4] for the frontend, the Spring framework[5] for the backend and Hibernate[6] as ORM framwork to achieve persistence. Multiple users can self register at one user agent protected by simple username/password authentication in a consequence (a more sophisticated mechanism might be desired for productive systems). After configuring the POP3 and SMTP server in the corresponding mail setting menu, the user agent allows to send and receive certified emails as well as regular emails. Furthermore, details of the transfer state are provided, i.e., if valid NRK, NRS and NRD evidences are available for sent certified emails, respectively a valid NRO evidence for received certified mails. In case the evidence is available also the corresponding timestamp is shown. If a transfer is canceled by timeout or for another reason on the other hand

[4] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[5] https://spring.io/

[6] http://hibernate.org/

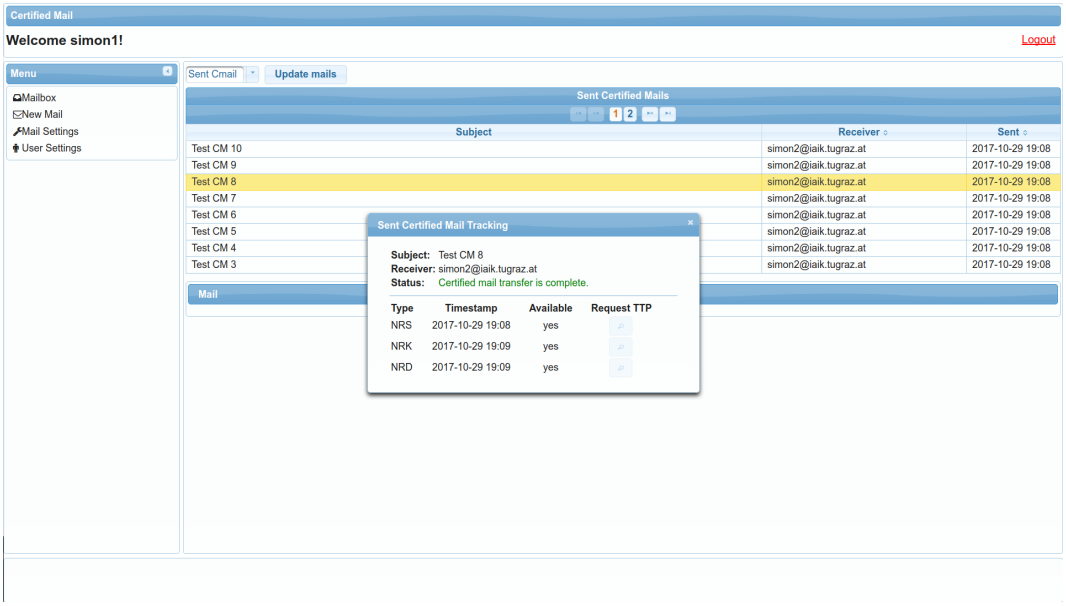

Figure 7.2.: A lightweight certified mail enabled user agent.

the user is informed as well.

7.4. Communication Protocols

The certified email system prototype is implemented on top of already existing protocols. SMTP is used to transfer messages from user agent to MTA, between MTAs and from the certified email service back to the MTA after message processing. The SMTP protocol requires no changes - it can be taken as it is. For the synchronization operation between user agent and the MTA on the other hand either POP3 or IMAP offer themselves, while POP3 was selected for the prototype implementation. Both potential protocols require minor changes: a timestamp has to be appended to the synchronization request and a signature has to be appended. For the prototype system those requirements are met by extending the Java mail client library. Request timestamp, signature key and meta-information about the signature algorithm have to be attached to the POP3 session instance properties. Those

properties are used in a consequence to extend the commands according the requirements before the command is issued. The following listing shows the format of the modified `RETR` command while ltsRQ is the timestamp in milliseconds and the signature is Base64 encoded:

```
RETR + messageNumber + ltsRQ + signature
```

As operational communication protocol between the TTP and corresponding entities Java RMI is employed. The same protocol is used for internal forwarding of certified email messages from the MTA to the certified email service. While this protocol fits ideal for the prototype system a different approach might be desired for a productive system, especially for the TTP communication (see 10).

7.5. Trusted Third Party

The trusted third party is a Spring based Java application accepting certified email resolution requests. Incoming information attached to the requests is persisted using Hibernate to be available for potential later requests. If necessary, depending on the request and already persisted data, the TTP is able to contact MTAs, respectively the corresponding certified email service to request evidences and additional data using Java RMI. MTA responses are persisted similar to request attributes to be used for potential later requests and if an evidence is available after a response it is returned to the requesting entity. If the TTP is not able to reach certain entities required for resolution, it is able to generate affidavit evidences if enough information is already available from previous requests. Otherwise, if the time-limit is overdue or if no resolution is possible the TTP might either answer with a signed `nack` command or it will cancel the delivery depending on the resolution protocol state. The prototype certified email TTP implementation additionally validates the signatures of all incoming requests and responses, respectively the validity of collected evidences to ensure a valid system status. Special attention has to be paid to synchronization: As every resolution request is started as a new thread, it is potentially possible that resolution is requested for one distinct certified email from different entities. In order to mitigate undefined transfer states, a synchronization mechanism

is integrated in the TTP prototype preventing parallel resolution for one distinct certified email - different mails can still be resolved concurrently. Furthermore, the prototype system is also enhanced with functionality to provide synchronized timestamps and a public key server.

- Timestamp server: Parties can request a reference time of this server. Timing is essential as the certified email system depends on a synchronized time.
- Public key server: The public key server allows parties to register their public keys and make them publicly available for other parties in a consequence. The prototype certified email system registers a public key implicitly when a new email account is created while the key is bound to the corresponding email address. Other parties may request a public key by a request providing the corresponding email address afterwards.

7.6. Message Format

In order to transfer certified email messages an underlying message format has to be chosen or defined. For the certified email prototype implementations JWS (JSON Web Signature) objects are used as they fit ideal for this purpose. JWS format allows structuring data and adding signatures conveniently while keeping metadata short and simple compared to XML or ASN1 for example. For transfer the serialized JWS object is attached as payload to an email of the newly defined MIME type *multipart/certified*.

8. Certified Email Forwarding

In Section 3 analogues certified mail systems were compared with regular email technology in order to identify the lacks to be fulfilled by certified electronic mail systems in Section 3.3. Based on those properties a certified mail system fulfilling those requirements was implemented on top of already existing Internet email technology. Nevertheless, when comparing the resulting system with services offered by analogous certified mail providers there is still a functionality missing, as cited from Section 3.1.1: *"Optionally a sender can also determine whether only the intended recipient itself or optionally also an authorized person may take the delivery"*.

If certified email systems should be used in the business sector for instance, mail forwarding is a substantial feature. A scenario demonstrating the importance of mail forwarding could be an employee on vacation. Often it is unavoidable to process the received business emails addressed to the person in absence nonetheless. As the implemented certified email system provides end-to-end encryption, the private key of the recipient is necessary to decrypt the message, hence it cannot be simply forwarded to a replacement person. In order to get access to the private key the absent person would have to share the private key with others or a system administrator would have to intervene. Summarized, those solutions are not desirable.

8.1. A possible Solution for Certified Email Forwarding

The *delivery sub-protocol* of the implemented certified email system consists of two main phases (6): In the first phase the symmetrical encrypted ciphertext containing the message content is transferred while at this time the

symmetric key for decryption is unknown to the recipient. In the second phase, in exchange of a NRK evidence, the symmetric key is transferred to the recipient while this key is encrypted with a traditional public key encryption scheme. For encryption the recipient's public key is applied, hence it can only be decrypted by provision of the recipient's private key. Within the last years a comparably new encryption technology referred to as proxy re-encryption gained more and more popularity. By employing this technology and extending the implemented protocol a certified mail forwarding solution can be realized.

8.1.1. Proxy re-encryption

The proxy re-encryption primitive was initially introduced by Blaze et al. in 1998 [48]. As this scheme has several drawbacks and considerable security risks it is not very applicable for practical usage scenarios. Nevertheless, the idea was adopted and the scientific community developed several new and improved proxy re-encryption schemes based on elliptic curves and lattices providing different properties, each. Regular public key encryption schemes allow transferring data to a distant party without the need for exchanging a secret key. The following set of functions is available:

$$\text{Keygen}(\lambda) \rightarrow (sk_B, pk_B) \tag{8.1}$$

$$\text{Encrypt}(M, pk_B) \rightarrow (C_B) \tag{8.2}$$

$$\text{Decrypt}(C_B, sk_B) \rightarrow (M) \tag{8.3}$$

By providing some encryption primitive environment parameters λ, Keygen allows to generate a keypair consisting of a public key pk_B, respectively a private key sk_B . When data should be transferred encrypted from user **A** to user **B**, the former uses Encrypt to encrypt a message M with the recipients public key pk_B. After transferring the encrypted data C_B, user **B** is able to reconstruct the plaintext M by decrypting the ciphertext using Decrypt under provision of the ciphertext C_B and **B**'s private key . Hereby the private key sk_B never leaves user **B**'s domain. Proxy re-encryption adds two further functions:

$$\text{ReKeygen}(sk_B, pk_C) \rightarrow (rk_{B\rightarrow C}) \tag{8.4}$$

$$\text{ReEncrypt}(C_B, rk_{B\rightarrow C}) \rightarrow (C_C) \tag{8.5}$$

If user **B** creates a re-encryption key $rk_{B\rightarrow C}$ by executing ReKeygen under the provision of a user **C**'s public key pk_C and **B**'s secret key sk_B, a third party can use this re-encryption key to transform ciphertext. More precisely, a third party can transform ciphertext C_B, decryptable by user **B**'s private key, into ciphertext C_C, decryptable by user **C**'s private key, after applying ReEncrypt. Hereby the re-encryption key can be classified as public key and the third party has at no time the possibility to access the plaintext.

8.1.2. Protocol Extension

It is possible to integrate a certified email forwarding service into the implemented certified email system without further modification of any standard email protocol (i.e. SMTP, POP3 or IMAP4) beyond the signature modifications necessary for the implemented certified email protocol anyway and without the need of major changes in the existing server side mailing software (i.e. MTA or MDA). Hereinafter a procedure for realizing certified email forwarding while maintaining the required certified email properties is presented. We assume a user **B** intending to activate forwarding to a user **C** due to unavailability.

1. User agent **B** generates a re-encryption key $rk_{B\rightarrow C}$ on behalf of the user.
2. The resulting key and the forwarding information, e.g., the replacement receiving email address and optionally a time-frame indicating the forwarding duration have to be transferred to the MTA where the user is registered. The transfer can be realized by sending the key embedded in an email over regular SMTP. For this email a specific MIME type or header has to be defined.
3. If the MTA detects an email including this header or MIME type it is responsible to forward it directly to the certified mail extension.
4. The certified mail extension stores the re-encryption key and forwarding information.

5. If user agent **A** intends to send a certified email to user agent **B** and forwarding is activated, the certified email extension will recognize this based on the stored forwarding information. In a consequence it will act as man in the middle for M3, M7 and M9 and forward those messages directly.
6. Forwarding M8 on the other hand requires some processing: Before M8 is forwarded the certified mail extension has to perform a re-encryption operation on $PK_{UA_R}[k]$ using the stored re-encryption key $rk_{B \to C}$.
7. In order to preserve the possibility to execute the resolution protocol, the certified mail extension on the original receiving side is responsible to answer with forwarding information in case it is contacted. Based on this information the TTP is able to contact the replacement certified mail extension and gain the necessary information from there. The forwarding information should be stored at the TTP after the first corresponding response.
8. If the forwarding should be stopped, user **B** has to inform the certified email extension by sending a stop signal wrapped into an email with the same MIME type or header used for starting.

Figure 8.1 demonstrates the forwarding of a certified email sent by user **A** from user **B** to user **C** as a sequence diagram.

8.2. Certified Email Forwarding Evaluation

The proposed solution enables to extend the implemented certified email service with mail forwarding capabilities while end-to end encryption and all other certified email requirements are maintained. Depending on the configured SMTP authentication mechanism, it is also imaginable to pre-compute re-encryption keys and use those keys to activate certified mail forwarding in situations where a user leaves unexpectedly. If the proxy re-encryption scheme is implemented using lattices a further advantage is obtained: Lattices are considered as post-quantum cryptography secure so far. As symmetric encryption used for encrypting the plaintext itself is considered as relatively little susceptible for quantum cryptography as well,

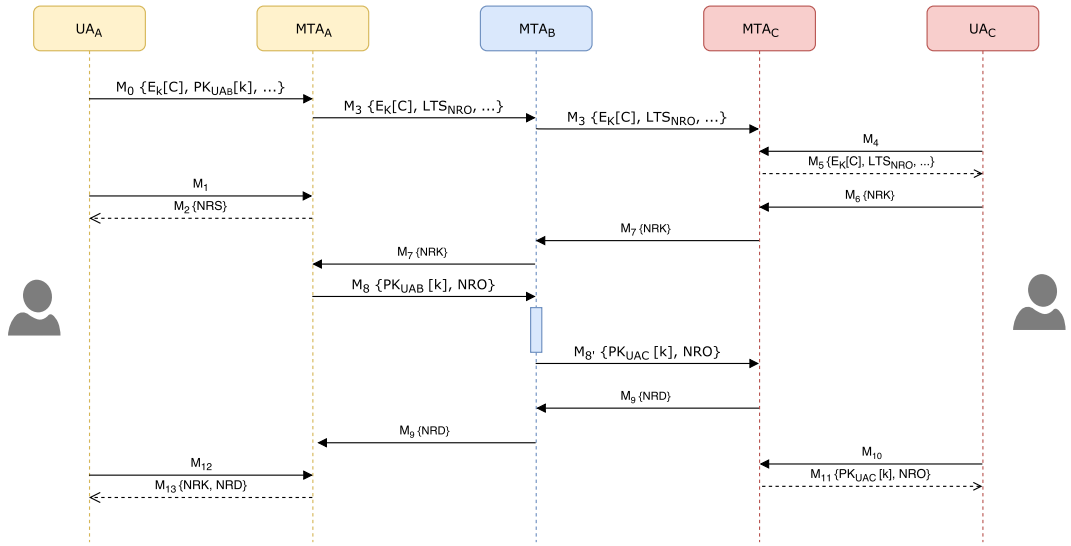

Figure 8.1.: Certified email forwarding sequence: User **A** sends a certified email to user **B**. **B** activated forwarding to user **C**. k: symmetric key, C: message, $E_k[C]$: symmetric encrypted message, PK_{UA_B}: public key of user agent B, $PK_{UA_B}[k]$: symmetric key encrypted with public key of user agent B, LTS_{NRO}: NRO timestamp, M'_8: copy of M_8 while PK_{UA_B} is re-encrypted to PK_{UA_C} by MTA **B**.

a system potentially able to withstand quantum cryptography would be the result.

9. Evaluation

An electronic certified email system on top of basic Internet email protocols fulfilling all requirements (3.3) to obtain an adequate replacement for analogous certified mail was implemented. Furthermore, the protocol was extended by certified mail forwarding functionality using proxy re-encryption - an important feature especially in business context. This section presents observations made during the implementation concerning the certified email protocol itself next to observations concerning the implemented components. Suggestions for improvements based on this evaluation results are presented in the future work section (10).

9.1. Protocol Evaluation

In general the protocol is able to fulfill all defined requirements excellently. Nevertheless, some things have to be kept in mind:

- For the implemented protocol it is necessary to sign the user agents synchronization commands. This is especially important to prevent repudiation of knowledge when receiving a certified email. In order to implement this feature, the protocol authors suggest developing POP3 extensions for signing `LIST` and `RETR` commands, respectively a similar extension for IMAP. Nevertheless, solving this requirement using extensions faces some problems:
  - Instead of only signing `LIST` and `RETR` commands **all** commands exposing information about new emails in the inbox (including metadata) have to be signed (e.g. `TOP` in case of POP3 and many more in case of IMAP4) and recorded by the MTA. This problem

increases as MTAs can deploy further extensions exposing information about new emails; therefore a policy would be necessary for certified email MTAs enforcing only to allow the deployment of such extensions that enforce signatures on information exposing client commands.
  - Extensions implemented according the protocol extension mechanism of POP3 2.3.2 respectively IMAP 2.4.2 introduce new commands. This does not meet the requirements for certified email as the default unsigned commands still could be executed. For certified email the existing commands have to be enhanced with signatures or signed extensions have to be developed while the default commands must be deactivated resulting in incompatibility with other participants. Alternatively fundamental changes have to implemented in the MTA preventing exposure of unprivileged information using commands not adequate for certified email.

- If any party requested a successful TTP resolution and the delivery sub-protocol is restarted by this party in a consequence, the protocol implementation must be able to detect and prevent potential message duplication.
- In order to obtain the best resolution results in perspective of performance and to reduce unnecessary communication effort between TTP and other parties, special attention should be paid on the configurable time limits defining the start of the resolution process. While reasonable values are pre-configured in the prototype applications, it might be advantageous to adjust those values when gaining more experience in real-world deployment and usage.

9.2. Implementation Evaluation

Beside the protocol evaluation the implementation of the server, client and third party components is evaluated below:

9.2.1. MTA and Certified Email Service

By choosing the approach to outsource certified mail processing from the MTA to an independent certified mail service application a very flexible, scalable and reusable solution could be realized on the server-side. Next to validating synchronization request signatures, the only additional task to be accomplished by the MTA itself is to identify certified email messages based on the MIME type and forward those messages to the certified mail service for further processing. In a consequence it is possible for email providers to offer certified mail services by making only minor changes in the MTA software source code. No new databases or additional substantial dependencies are required in the MTA software making it also possible to integrate certified email in sophisticated, optimized, C or other low level programming language MTAs with minor effort. The certified mail service application on the other hand has common requirements for typical applications: a database and a signature key-pair bound to the application.

9.2.2. Client

Basically certified email functionality can be integrated in every already existing mail client software. Nevertheless, the modifications to be done are more extensive compared to the MTA software. Changes in the user interface are necessary, additional database tables have to be created, additional dependencies have to be provided (e.g. proxy re-encryption library when certified mail forwarding is included), key-pairs for signatures and encryptions have to be maintained and a periodical task runner for starting the resolution process if necessary has to be implemented. Therefore, in order to implement the prototype certified mail protocol without facing major problems resulting from extension limitations of already existing software a new, lightweight certified mail user agent was developed. If certified email functionality should be integrated in existing UA software in the future, the implemented prototype will be very helpful, as a system for comparing and testing by replacing desired components is available. Another interesting facet regarding the client software is the type of the application: The resolution protocol is usually started before the certified

email transfer times out. This is no problem if the UA is realized as always on-line web-application employing a task runner for all clients. Nevertheless, if the UA is realized as native application, similar to Thunderbird or Microsoft Outlook, the start of the resolution protocol in time cannot be guaranteed as the hosting computer or the application can be turned off.

9.2.3. TTP, Time Synchronization and PKI

For the present certified email implementation the TTP was equipped with a reference time provider and a simple key-server for all certified email system participants replacing a sophisticated PKI solution, as this is out of scope for the prototype. As key and especially time-stamp queries lead to lot of traffic between TTP and other entities those components should be split up and enhanced (functionalities provided by sophisticated PKI solutions should be installed, e.g., revocation mechanisms) for a productive system. By implementing the TTP particular attention was payed to synchronization: If multiple entities start certified mail resolution in the same point of time this could lead to problems otherwise, as different entities could gain different information about the certified mail transfer state.

10. Future Work

The implemented certified email system serves in first instance the purpose to provide a prototype solution demonstrating the feasibility of integrating a certified email system on top of the current Internet email infrastructure. With Java a high programming language was chosen making it possible to conveniently include additional modifications for testing purposes. If the intention is on the other hand to develop a system for real-world usage, some enhancements should, respectively can be implemented. Those improvements wouldn't have put additional benefits to the prototype as it is out of scope:

- **Protocols:** Currently the communication between the TTP and other parties, respectively the communication from Apache James to the certified mail service is realized using the Java RMI protocol. While this protocol is feasible for a prototype implementation, a more generic protocol should be consulted for productive usage for two reasons:
  - JAVA RMI is very platform dependent, it works only for Java applications.
  - If the communicating parties are not on the same host being usually the case in a real-world environment, RMI requires opening of some specific ports. Furthermore, RMI is not really designed for communication over the Internet.

  A better and very flexible solution for a real-world system could for example be to use JSON messages over plain sockets. For the communication from the MTA to the certified mail service also LMTP would offer itself.
- **Authentication:** For providing a secure certified email system, proper authentication is mandatory between user agent and mail transfer agent. The prototype employs only simple username and password

authentication. This authentication mechanism is not adequate for certified mails. Instead, more secure solutions should be installed, for example two-factor authentication.

- **MIME type registration:** New MIME types were introduced to implement the certified mail system prototype. For productive usage those MIME types should be officially registered according 2.5.2.
- **IMAP:** The prototype implementation currently supports only POP3 for the message synchronization, as enabling IMAP4 support is rather complex. Nevertheless, for productive usage this might be a feature of interest. If IMAP4 support should be added some things have to be kept in mind:
  - **All** IMAP commands disclosing information about received certified emails (also meta information) have to be signed and recorded by the certified mail service in order to prevent selective reception.
  - As IMAP4 is intended to be used by multiple clients simultaneously, it has to be take care that no synchronization problems occur in this case.
  - Emails in IMAP4 can be distributed over several mailboxes and they are usually not deleted on the server-side.
- **Public key infrastructure (PKI):** Instead of a sophisticated PKI only a simple key-server was used to realize the prototype. This simple system does not provide functionalities necessary to operate a productive certified email system. There is no possibility for public key chain validation or revocation for example. Therefore, for a real world system, a more sophisticated public key infrastructure should be installed. On the protocol level support for PKI is already possible as JWS is used as message container providing several options to pass necessary information.
- **Timestamps:** For a certified email system it is necessary that all participating entities have correct time values - this is even more essential if UAs are realized as native client applications instead of web applications. For the prototype this was realized by using the TTP time as reference time. In order to reduce the communication with the TTP server a different reference time solution might be installed.
- **Multiple recipients:** Currently the implemented certified email pro-

totype is only able to handle certified emails with a single recipient. Certified email providers might also be interested in sending certified mails to multiple recipients. This can be realized without any protocol modifications by some minor modifications in the UA implementation. For each recipient the certified email should be copied and a new transaction should be initiated.

Bibliography

[1] D. Spicer. "Raymond Tomlinson: Email Pioneer, Part 1." In: *IEEE Annals of the History of Computing* 38.2 (Apr. 2016), pp. 72–79. ISSN: 1058-6180. DOI: 10.1109/MAHC.2016.25 (cit. on p. 2).

[2] Richard W. Watson. *A MAIL BOX PROTOCOL*. July 1971. URL: https://tools.ietf.org/html/rfc196 (visited on 11/26/2016) (cit. on p. 2).

[3] C. Partridge. "The Technical Development of Internet Email." In: *IEEE Annals of the History of Computing* 30.2 (Apr. 2008), pp. 3–29. ISSN: 1058-6180. DOI: 10.1109/MAHC.2008.32 (cit. on p. 3).

[4] David H. Crocker. *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. RFC 822. Dept. of Electrical Engineering, Aug. 1982, pp. 1–47. URL: https://www.ietf.org/rfc/rfc0822.txt (cit. on p. 3).

[5] P. Resnick. *Internet Message Format*. RFC 5322. Qualcomm Incorporated, Oct. 2008, pp. 1–57. URL: https://tools.ietf.org/html/rfc5322 (cit. on pp. 3, 20–22).

[6] J. K. Reynolds. *POST OFFICE PROTOCOL*. RFC 918. Oct. 1984, pp. 1–5. URL: https://tools.ietf.org/html/rfc918 (cit. on p. 3).

[7] M. Crispin. *INTERACTIVE MAIL ACCESS PROTOCOL - VERSION 2*. RFC 1064. July 1988, pp. 1–26. URL: https://tools.ietf.org/html/rfc1064 (cit. on p. 3).

[8] Craig Partridge. *MAIL ROUTING AND THE DOMAIN SYSTEM*. RFC 974. CSNET CIC BBN Laboratories Inc, Jan. 1986, pp. 1–7. URL: https://tools.ietf.org/html/rfc974 (cit. on p. 4).

[9] J. Klensin et al. *SMTP Service Extension for 8bit-MIME Transport*. RFC 1426. Feb. 1993, pp. 1–6. URL: https://tools.ietf.org/html/rfc1426 (cit. on p. 4).

[10] Inc The Radiacti Group. *Email Statistics Report, 2016-2020*. Mar. 2016. URL: http://www.radicati.com/wp/wp-content/uploads/2016/01/Email_Statistics_Report_2016-2020_Executive_Summary.pdf (visited on 12/10/2016) (cit. on p. 4).

[11] SecuritySpace. *Mail (MX) Server Survey*. Jan. 2017. URL: http://www.securityspace.com/s_survey/data/man.201612/mxsurvey.html (visited on 12/10/2016) (cit. on p. 4).

[12] *Open email survey*. Mar. 2016. URL: http://www.openemailsurvey.org/ (visited on 12/10/2016) (cit. on p. 4).

[13] Litmus Labs. *2016 State of Email Report*. Mar. 2016. URL: https://litmus.com/blog/we-analyzed-13-billion-opens-to-discover-where-subscribers-read-email-infographic (visited on 01/08/2017) (cit. on p. 4).

[14] D. Crocker. *Internet Mail Architecture*. RFC 5598. July 2009, pp. 1–54. URL: https://tools.ietf.org/html/rfc5598 (cit. on p. 7).

[15] J. Klensin R. Gellens. *Message Submission for Mail*. RFC 6409. Nov. 2011, pp. 1–20. URL: https://tools.ietf.org/html/rfc6409 (cit. on p. 8).

[16] J. Klensin. *Simple Mail Transfer Protocol*. RFC 5321. Oct. 2008, pp. 1–95. URL: https://tools.ietf.org/html/rfc5321 (cit. on pp. 10, 12, 14).

[17] K. Moore. *Simple Mail Transfer Protocol (SMTP) Service Extension for Delivery Status Notifications (DSNs)*. RFC 3461. Jan. 2003, pp. 1–38. URL: https://tools.ietf.org/html/rfc3461 (cit. on pp. 14, 45).

[18] J. Myers et al. *Post Office Protocol - Version 3*. RFC 1939. May 1996, pp. 1–23. URL: https://www.ietf.org/rfc/rfc1939.txt (cit. on p. 15).

[19] R. Gellens et al. *POP3 Extension Mechanism*. RFC 2449. Nov. 1998, pp. 1–19. URL: https://tools.ietf.org/html/rfc2449 (cit. on p. 17).

[20] M. Crispin. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501. Mar. 2003, pp. 1–108. URL: https://tools.ietf.org/html/rfc3501 (cit. on pp. 17, 18).

[21] N. Borenstein N. Freed. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045. Nov. 1996, pp. 1–31. URL: https://tools.ietf.org/html/rfc2045 (cit. on p. 23).

[22] N. Borenstein N. Freed. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046. Nov. 1996, pp. 1–44. URL: https://tools.ietf.org/html/rfc2046 (cit. on pp. 23, 24).

[23] K. Moore. *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*. RFC 2047. Nov. 1996, pp. 1–15. URL: https://tools.ietf.org/html/rfc2047 (cit. on pp. 23, 24).

[24] J. Klensin N. Freed. *Media Type Specifications and Registration Procedures*. RFC 4288. Dec. 2005, pp. 1–24. URL: https://tools.ietf.org/html/rfc4288 (cit. on pp. 23, 25).

[25] J. Klensin N. Freed. *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*. RFC 4289. Dec. 2005, pp. 1–11. URL: https://www.ietf.org/rfc/rfc4289.txt (cit. on pp. 23, 26).

[26] N. Borenstein N. Freed. *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*. RFC 2049. Nov. 1996, pp. 1–24. URL: https://tools.ietf.org/html/rfc2049 (cit. on p. 23).

[27] K. Zeilenga A. Melnikov. *Simple Authentication and Security Layer (SASL)*. RFC 4422. June 2006, pp. 1–33. URL: https://tools.ietf.org/html/rfc4422 (cit. on p. 27).

[28] P. Hoffman. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207. Feb. 2002, pp. 1–9. URL: https://www.ietf.org/rfc/rfc3207.txt (cit. on p. 28).

[29] C. Newman. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595. June 1999, pp. 1–15. URL: https://tools.ietf.org/html/rfc2595 (cit. on p. 28).

[30] J. Galvin et al. *Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted*. RFC 1847. Oct. 1995, pp. 1–11. URL: https://tools.ietf.org/html/rfc1847 (cit. on p. 29).

[31] M. Elkins et al. *MIME Security with OpenPGP*. RFC 3156. Aug. 2001, pp. 1–15. URL: https://tools.ietf.org/html/rfc3156 (cit. on p. 29).

[32] S. Turner B. Ramsdell. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751. Jan. 2010, pp. 1–45. URL: https://tools.ietf.org/html/rfc5751 (cit. on p. 30).

[33] Arne Tauber. "Cross-border Certified Electronic Mailing: A Scalable Interoperability Framework for Certified Mail Systems." PhD thesis. 2012 (cit. on pp. 35, 37, 42, 46).

[34] Joseph Lluis Ferrer-Gomilla et al. "Certified electronic mail: properties revisited." In: *Computers & Security* 29 (Apr. 2010), pp. 167–179. ISSN: 0167-4048 (cit. on p. 35).

[35] K. Moore. *Message Disposition Notification*. RFC 3798. May 2004, pp. 1–30. URL: https://tools.ietf.org/html/rfc3798 (cit. on p. 45).

[36] A. Melnikov. *Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)*. RFC 3503. Mar. 2003, pp. 1–9. URL: https://tools.ietf.org/html/rfc3503 (cit. on p. 45).

[37] P. Hoffman. *Enhanced Security Services for S/MIME*. RFC 2634. June 1999, pp. 1–58. URL: https://www.ietf.org/rfc/rfc2634.txt (cit. on p. 45).

[38] Jianying Zhou and Dieter Gollmann. "Certified electronic mail." In: *Computer Security — ESORICS 96: 4th European Symposium on Research in Computer Security Rome, Italy, September 25–27, 1996 Proceedings*. Ed. by Elisa Bertino et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 160–171. ISBN: 978-3-540-70675-5. DOI: 10.1007/3-540-61770-1_35. URL: https://doi.org/10.1007/3-540-61770-1_35 (cit. on p. 46).

[39] Jianying Zhou and Dieter Gollmann. "A Fair Non-repudiation Protocol." In: *Proceedings of the 1996 IEEE Conference on Security and Privacy*. SP'96. Oakland, California: IEEE Computer Society, 1996, pp. 55–61. ISBN: 0-8186-7417-2. URL: http://dl.acm.org/citation.cfm?id=1947337.1947348 (cit. on p. 46).

[40] Martín Abadi and Neal Glew. "Certified email with a light on-line trusted third party: Design and implementation." In: (Jan. 2002), pp. 387–395 (cit. on p. 46).

[41] Jianying Zhou and Dieter Gollmann. "An Efficient Non-repudiation Protocol." In: *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*. CSFW '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 126–. ISBN: 0-8186-7990-5. URL: http://dl.acm.org/citation.cfm?id=794197.795082 (cit. on p. 46).

[42] Guilin Wang. "Generic Fair Non-Repudiation Protocols with Transparent Off-line TTP." In: *Proceedings of the 2005 Conference on Applied Public Key Infrastructure: 4th International Workshop: IWAP 2005*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2005, pp. 51–65. ISBN: 1-58603-550-9. URL: http://dl.acm.org/citation.cfm?id=1564104.1564111 (cit. on p. 46).

[43] H. Wang et al. "A New Certified Email Protocol." In: *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)*. Sept. 2007, pp. 683–687. DOI: 10.1109/DEXA.2007.117 (cit. on p. 46).

[44] H. Wang et al. "Certified Email Delivery with Offline TTP." In: *Third International Symposium on Information Assurance and Security*. Aug. 2007, pp. 15–20. DOI: 10.1109/IAS.2007.85 (cit. on p. 46).

[45] Carlo Blundo, Stelvio Cimato, and Roberto De Prisco. "Certified Email: Design and Implementation of a New Optimistic Protocol." In: *Proceedings of the Eighth IEEE Symposium on Computers and Communications (ISCC 2003), 30 June - 3 July 2003, Kiris-Kemer, Turkey*. 2003, pp. 828–833. ISBN: 0-7695-1961-X. DOI: 10.1109/ISCC.2003.1214220. URL: http://dx.doi.org/10.1109/ISCC.2003.1214220 (cit. on p. 46).

[46] D. Liu et al. "A Practical Certified E-Mail System with Temporal Authentication Based on Transparent TSS." In: *2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. Aug. 2008, pp. 285–290. DOI: 10.1109/SNPD.2008.119 (cit. on p. 46).

[47] G. Draper-Gil et al. "An optimistic certified e-mail protocol for the current Internet e-mail architecture." In: *2014 IEEE Conference on Communications and Network Security*. Oct. 2014, pp. 382–390. DOI: 10.1109/CNS.2014.6997507 (cit. on pp. 47, 49, 53–55).

[48] Matt Blaze, Gerrit Bleumer, and Martin Strauss. "Divertible Protocols and Atomic Proxy Cryptography." English. In: *Advances in Cryptology — EUROCRYPT'98*. Vol. 1403. Lecture Notes in Computer Science. 1998, pp. 127–144. ISBN: 978-3-540-64518-4 (cit. on p. 68).

Appendix

Appendix

A. Email Server Software

Table .1 compares desired features of server-side email software including the programming language, operating system support, license (proprietary or open source) and the protocols included while a list targeting the comparison of mail servers published on wikipedia was used as basis [1].

Name	Lang	Linux	Lic	Features
agorum core	Java	yes	os	SMTP/IMAP
Apache James	Java	yes	os	SMTP/IMAP/POP3
Atmail	PHP	yes	dual	SMTP/IMAP/POP3
Axigen	?	yes	prop	SMTP/IMAP/POP3
Citadel	C	yes	os	SMTP/IMAP/POP3
Cloudmark	?	yes	prop	SMTP
CommuniGate Pro	?	yes	prop	SMTP/IMAP/POP3
Courier Mail Server	C/C++	yes	os	SMTP/IMAP/POP3
Cyrus IMAP server	C	yes	os	IMAP/POP3
Dovecot	C	yes	os	IMAP/POP3
Eudora Internet Mail Server	?	no	prop	SMTP/IMAP/POP3
Exim	C	yes	os	SMTP
FirstClass	?	yes	prop	SMTP/IMAP/POP3
Gordano Messaging Suite	?	yes	prop	SMTP/IMAP/POP3
GroupWise	?	yes	prop	SMTP/IMAP/POP3
Halon	C++	yes	prop	SMTP
Haraka	Node.js	yes	os	SMTP
hMailServer	C++	no	os	SMTP/IMAP/POP3
IBM/Lotus Notes/Domino	Java/C++	yes	prop	SMTP/IMAP/POP3
IceWarp Mail Server	?	yes	prop	SMTP/IMAP/POP3
Ipswitch IMail Server	?	no	prop	SMTP/IMAP/POP3
Kolab	C++/PHP	yes	os	SMTP/IMAP/POP3
Kopano	C++/PHP	yes	os	SMTP/IMAP/POP3
Mailsite	?	no	prop	SMTP/IMAP/POP3
Mailtraq	?	no	prop	SMTP/IMAP/POP3
MDaemon	?	no	prop	SMTP/IMAP/POP3

[1] https://en.wikipedia.org/wiki/Comparison_of_mail_servers

Mercury Mail Transport	?	no	prop	SMTP/IMAP/POP3
Microsoft Exchange Server	C++/C#	no	prop	SMTP/IMAP/POP3
NetMail	?	yes	prop	SMTP/IMAP/POP3
OpenSMTPD	C	yes	os	SMTP
Open-Xchange	C/Java	yes	dual	SMTP/IMAP/POP3
Oracle Communications	?	yes	prop	SMTP/IMAP/POP3
Postfix	C	yes	os	SMTP
Qmail	C	yes	os	SMTP/POP3
Qpopper	C	yes	os	POP3
Sendmail	C	yes	os	SMTP
SparkEngine	?	yes	prop	SMTP
Synovel Collabsuite	?	yes	prop	SMTP/IMAP/POP3
UW IMAP	C/C++	yes	os	IMAP/POP3
WinGate	?	no	prop	SMTP/IMAP/POP3
Zarafa	C++	yes	os	SMTP/IMAP/POP3

Table .1.: Desired features for email server software. Abbreviations: Lang - Programming Language, Linux - Linux support, Lic - Licence, os - open source, prop - proprietary, Features - protocols provided.

B. Demo Environment Setup

This section provides an example how to set up a demo environment similar to figure 7.1. In order to come close to a real world scenario this setup contains two independent email servers including the certified mail extension services, two independent email clients and one common trusted third party (TTP). In order to omit the requirement of setting up a DNS server for simplification, the environment is configured only for particular demo domains. Embedded derby databases are used per default while it is possible and convenient to configure several other database types as well. It has to be taken care that the trusted third party should be running before starting the certified mail server extensions, as the latter generates a signature keypair after starting followed by a deployment of the public signature key at the TTP instance. The following instruction leads

from source code to be compiled to a running environment including the necessary configuration. In order to shorten the compiling time the flag *-DskipTests=true* can be used for the maven commands. For inspecting the derby databases the application DBeaver has proven itself.

B.1. Modified javax.mail Library

The modification of the javax.mail library was arranged as maven project. There is no configuration necessary. The only thing to be considered here is to compile the library as well as installing it to the local maven repository as other projects make use of it. Both tasks can be achieved with the following command:

```
mvn install
```

B.2. Trusted Third Party

The TTP was arranged as maven project. In order to compile and generate a runnable jar file the following command should be executed:

```
mvn package
```

Herby a jar file is created in the target folder that can be copied to the desired destination. The database schemas and files can be generated automatically in a consequence. Therefore the file *applicationContext.xml* inside the jar file has to be changed. Particularly the property *hibernate.hbmdll.auto* has to be altered from *validate* to *create*:

```
<beans:prop key="hibernate.hbm2ddl.auto">create</beans:prop>
```

Afterwards the database can be created by:

```
java -jar cmailTTP.jar
```

This command creates the database and the schemas relative to the jar file in the same directory. When the schemas were created successfully the TTP should be shut down again followed by changing back *create* to *validate*

inside the *applicationContext.xml* file. For finally starting up the TTP the following command can be used:

```
nohup java -jar cmailTTP.jar &
```

It has to be paid attention that this way the task has to be killed manually for shutting the TTP down again.

B.3. Apache James

Apache James is a maven project as well. For compilation and package generation the following command should be used (the *-DSkipTests=true* flag is highly recommended here as unit tests take a lot of time otherwise):

```
mvn package -DskipTests=true -Pwith-assembly
```

Unfortunately the RMI port to the certified mail extension service is still hardcoded currently making it necessary to compile at least the corresponding classes twice for obtaining two server instances. Those port values occur in two classes: *CertifiedMail.java* and *RetrCmdHandler.java*. Example values could be 1097 and 1098 (1099 should not be used in order to omit troubles). The resulting zip file is located in *james-project/server/app/target*. After unzipping to the desired destination several configuration work is necessary for each instance. The following builds on the demo default configuration coming with Apache James and has to be performed for both instances by adapting the respective attibutes. First of all several mailets have to be configured in *mailetcontainer.xml*. For activating certified email technology the corresponding mailet together with it's matcher have to be configured inside the *transport* processor configuration:

```
<mailet match="IsCertifiedMail" class="CertifiedMail" />
```

In order to enable sending mails in arbitrary domains without setting up a DNS server the following lines should be added to the *transport* processor:

```
<mailet match="HostIs=iaik1.tugraz.at"
 class="RemoteDelivery">
  <outgoingQueue>relay</outgoingQueue>
  <delayTime>5000, 100000, 500000</delayTime>
  <maxRetries>25</maxRetries>
```

```
    <maxDnsProblemRetries>0</maxDnsProblemRetries>
    <deliveryThreads>10</deliveryThreads>
    <sendpartial>true</sendpartial>
    <bounceProcessor>bounces</bounceProcessor>
    <gateway>0.0.0.0</gateway>
    <gatewayPort>2525</gatewayPort>
</mailet>
<mailet match="HostIs=iaik2.tugraz.at"
  class="RemoteDelivery">
    <outgoingQueue>relay2</outgoingQueue>
    <delayTime>5000, 100000, 500000</delayTime>
    <maxRetries>25</maxRetries>
    <maxDnsProblemRetries>0</maxDnsProblemRetries>
    <deliveryThreads>10</deliveryThreads>
    <sendpartial>true</sendpartial>
    <bounceProcessor>bounces</bounceProcessor>
    <gateway>0.0.0.0</gateway>
    <gatewayPort>2526</gatewayPort>
</mailet>
```

The domain name of the particular server has to be configured in *domainlist.xml*, for example:

```
<domainname>iaik1.tugraz.at</domainname>
```

The POP3 server can be configured in *pop3server.xml* in the following manner while a default keystore is used to be exchanged for productive usage:

```
<pop3servers>
  <pop3server enabled="true">
    <jmxName>pop3server</jmxName>
    <bind>0.0.0.0:8143</bind>
    <connectionBacklog>200</connectionBacklog>
    <tls socketTLS="false" startTLS="true">
      <keystore>file:./../conf/keystore</keystore>
      <secret>demopw</secret>
      <provider>
        org.bouncycastle.jce.provider.BouncyCastleProvider
      </provider>
    </tls>
    <connectiontimeout>1200</connectiontimeout>
    <connectionLimit>0</connectionLimit>
    <connectionLimitPerIP>0</connectionLimitPerIP>
    <handlerchain>
```

```
      <handler class="org.apache.james.pop3server.core.CoreCmdHandlerLoader"/>
    </handlerchain>
  </pop3server>
</pop3servers>
```

The SMTP server can be configured in *smtpserver.xml* in the following manner while a default keystore is used to be exchanged for productive usage:

```
<smtpservers>
  <smtpserver enabled="true">
    <jmxName>smtpserver-authenticated</jmxName>
    <bind>0.0.0.0:2525</bind>
    <connectionBacklog>200</connectionBacklog>
    <tls socketTLS="false" startTLS="true">
      <keystore>file://../conf/keystore</keystore>
      <secret>demopw</secret>
      <provider>
        org.bouncycastle.jce.provider.BouncyCastleProvider
      </provider>
      <algorithm>SunX509</algorithm>
    </tls>
    <connectiontimeout>360</connectiontimeout>
    <connectionLimit>0</connectionLimit>
    <connectionLimitPerIP>0</connectionLimitPerIP>
    <authRequired>announce</authRequired>
    <authorizedAddresses>0.0.0.0/0</authorizedAddresses>
    <verifyIdentity>true</verifyIdentity>
    <maxmessagesize>0</maxmessagesize>
    <addressBracketsEnforcement>true</addressBracketsEnforcement>
    <smtpGreeting>JAMES Linagora's SMTP awesome Server</smtpGreeting>
    <handlerchain>
      <handler class="org.apache.james.smtpserver.fastfail.ValidRcptHandler"/>
      <handler class="org.apache.james.smtpserver.CoreCmdHandlerLoader"/>
    </handlerchain>
  </smtpserver>
</smtpservers>
```

The jmx port used by the James client application has to be configured in *jmx.properties*:

```
jmx.address=127.0.0.1
jmx.port=9999
```

A derby database is generated in */var/store* automatically during the first startup without any further interaction. After the configuration is done the server can be started by executing *./james start* in the *bin* folder. In order to be operable a domain has to be created as well as james users (while the user *cmailservice* is required) using the james client in the *bin* directory:

```
sh james-cli.sh -h 127.0.0.1 -p 9999 adddomain
  iaik1.tugraz.at
sh james-cli.sh -h 127.0.0.1 -p 9999 adduser
  cmailservice@iaik1.tugraz.at cmailservice
sh james-cli.sh -h 127.0.0.1 -p 9999 adduser
  tom@iaik1.tugraz.at tom
```

A shutdown of the James server can be initiated by executing the following command in the bin folder:

```
sh james stop
```

B.4. Certified Mail Server Extension

Similar to former projects the Certified Mail Server Extension is arranged as maven project and can be compiled and packaged to a jar file by using the corresponding maven command:

```
mvn package
```

After copying the jar file to the desired destination the *config.properties* file in the jar file has to be adapted. The following listing demonstrates an example configuration:

```
mtaUrl = localhost
mtaPort = 2525
serviceMailAddress = cmailservice@iaik.tugraz.at
serviceMailPassword = cmailservice
rmiServerPort = 1098
signatureAlgorithm = SHA256withRSA
jwsSignatureAlgorithm = RS256
signatureKeyFormat = RSA
signatureKeyLength = 1024
```

The database can be created by changing the *hibernate.hbm2ddl.auto* property from *validate* to *create* inside *applicationContext.xml*:

```
<beans:prop key="hibernate.hbm2ddl.auto">create</beans:prop>
```

In order to initiate the creation of the database and the schema the Certified Mail Extensions should be started (and stopped again afterwards):

```
java -jar cmailService.jar
```

For starting the Certified Mail Extensions finally a truststore has to be provided containing a root certificate of James mail server. Additionally *hibernate.hbm2ddl.auto* has to be changed back to *validate*. The extensions can be started afterwards by the following command:

```
nohup java -jar -Djavax.net.ssl.trustStore="/opt/cmail/cmail1/ts"
 -Djavax.net.ssl.trustStorePassword="demopw"
 -Dorg.jboss.logging.provider=slf4j cmailservice-0.0.1-SNAPSHOT.jar &
```

This way the process has to be killed manually if a shutdown is desired.

B.5. Cmail Client

The last component to be deployed is the cmail user client. As this application is a web application, webapp runners, e.g. Apache Tomcat[2], have to be set up. The project itself is a maven project again and can be compiled and packaged by:

```
mvn package
```

Afterwards the resulting war file has to be copied to the according folder of the webapp runner. In order to create the database a folder for the database has to be created somewhere on the filesystem. The path to this folder has to be configured in the *applicationContext.xml* file located in the war file's WEB-INF folder:

```
<beans:property name="url" value="jdbc:derby:/opt/cmail/ua1/db/ua1;
  create=true" />
```

[2] https://tomcat.apache.org/

Additionally the *hibernate.hbm2ddl.auto* property in the same file should be changed from *validate* to *create* during the first startup. For the creation of the database the application should now be started and stopped again by executing the following in tomcat's bin folder:

```
sh startup.sh
sh shutdown.sh
```

For starting the application finally a truststore containing a root certificate of James server certificate has to be provided and configured. For the configuration a file named setenv.sh has to be created in tomcats bin folder containing the following content:

```
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=/opt/cmail/ua1/ts"
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStorePassword=demopw"
JAVA_OPTS="$JAVA_OPTS -Dorg.jboss.logging.provider=slf4j"
export JAVA_OPTS
```

Finally, after changing back the *hibernate.hbm2ddl.auto* property to *validate* the application can be started and stopped by:

```
sh startup.sh
sh shutdown.sh
```