

Mixtasy:
Remailing on Existing Infrastructure
Anonymized Email Communication Easily Deployable Using SMTP & OpenPGP

Master's Thesis

to gain the academic degree
Master of Science (M.Sc.)

at the
University of Applied Science FH Campus Wien
Department of Technology
Master's Degree Program: IT-Security

Submitted by Johannes Burk

Matriculation Number: 1410537029

1st Reviewer:

Dipl.-Inform. David Stezenbach

2nd Reviewer:

Priv.-Doz. Mag. DI. DI. Dr.techn. Karl Michael Göschka

Vienna, 21st of May 2016

Abstract

Email is one of the oldest electronic communication tools. While it provides excellent service in terms of simplicity and reliability, it suffers from high insecurities and does not provide any anonymity or privacy. Even the increasing use of known transport and end-to-end encryption techniques cannot avoid leaving a trace of metadata. Several existing solutions try to tackle these weaknesses, but non of them fit all requirements (including usability and adoption properties) regarding secure and anonymized communication.

Thus, this work develops “Mixtasy”, a new protocol based on a mix network. In order to enable rapid deployment, which facilitates better anonymization in this class of protocols, a central design goal was easy adoption. This is achieved by reusing existing technologies (OpenPGP, SMTP, Internet Message Format) and infrastructure (PGP key servers, mail transfer infrastructure). The practicability of the protocol is proven by a prototypical implementation. While excluding mixing algorithms and dummy traffic, the prototype offers basic anonymization of email communication. A complete implementation of the specification for protection against global adversaries is outstanding. Possible improvements, especially the optimization of the used mixing algorithm and enabling anonymous replies, are discussed and outlined for future work and research.

Kurzbeschreibung

E-Mails sind eines der ältesten elektronischen Kommunikationslösungen. Zwar ist die Kommunikation mit ihnen sehr einfach und zuverlässig, allerdings bieten sie keinerlei Schutz vor unbefugtem Mitlesen. Selbst die steigende Verwendung von bekannten Transport- und Ende-zu-Ende-Verschlüsselungstechniken kann das Hinterlassen von Metadaten nicht verhindern. Es gibt viele existierende Lösungen, die das Problem angehen. Aber keine erfüllt alle Anforderungen (inklusive Benutzerfreundlichkeit und Integrierbarkeit) bezüglich sicherer und anonymer Kommunikation.

Daher beschäftigt sich diese Arbeit mit der Entwicklung von “Mixtasy”, einem auf dem Konzept von Mix Networks basierendem Protokoll. Um eine rasche Verbreitung zu fördern, war eines der Hauptziele einfache Integrierbarkeit in bestehende Systeme. Dies wird durch das Wiederverwenden von existierenden Technologien (OpenPGP, SMTP, Internet Message Format) und Infrastruktur (PGP Key Server, E-Mail Server Infrastruktur) erreicht. Die Umsetzbarkeit wird mit einer prototypischen Implementierung bewiesen. Diese ermöglicht trotz dem noch nicht implementierten Mixing Algorithmus und der Nutzung von Dummy Traffic eine grundlegende Anonymisierung der E-Mail Kommunikation. Eine vollständige Umsetzung der Spezifikation für den Produktiveinsatz steht noch aus. Mögliche Verbesserungen, vor allem die Optimierung des verwendeten Mixing Algorithmus und das Ermöglichen anonymer Antworten, werden besprochen und für zukünftige Arbeiten und Forschung herausgestellt.

Contents

1	Introduction	6
2	Requirements and Objective	8
2.1	Threat Model	8
2.2	Security and Privacy	9
2.2.1	Conversation Security	9
2.2.2	Transport Privacy	9
2.3	Usability	10
2.4	Adoption	10
3	Technologies	11
3.1	Email	11
3.1.1	Internet Message Format	12
3.1.2	Simple Mail Transfer Protocol	12
3.1.3	Transport Encryption	13
3.1.4	End-to-End Encryption	14
3.2	Mix Networks and Onion Routing	15
4	Related Work	17
4.1	Remailer	17
4.1.1	Pseudonymous Remailer	18
4.1.2	Cypherpunk Remailer	18
4.1.3	Mixmaster	19
4.1.4	Mixminion	20
4.2	Dark Mail	21
4.3	Bitmessage	21
4.4	OnionMail	22
4.5	Reasons for a new Protocol	22

5	Design Considerations	24
5.1	Known Attacks to Mix Networks	24
5.1.1	Passive Attacks	24
5.1.2	Active Attacks	24
5.2	Mixing	25
5.3	Message Size	27
5.4	Replies and Sender Anonymity	27
5.5	Mix Discovery	28
5.6	Replay Attack Prevention	28
5.7	Tagging Attack Prevention	29
5.8	Dummy Traffic	29
5.9	Abuse	30
5.9.1	Opt-out Mechanism	30
5.9.2	Spam Resistance	30
6	Protocol Specification	32
6.1	Introduction	32
6.1.1	Terms	32
6.1.2	General Functions	33
6.1.3	Overall Design	33
6.2	Message Format	35
6.2.1	Original Message	37
6.2.2	Final Mix Message	37
6.2.3	Intermediate Mix Message	38
6.2.4	Header Fields	39
6.3	Key Management and Directory Service	41
6.3.1	Key Generation	41
6.3.2	Key Distribution and Discovery	42
6.3.3	Ranking	42
6.4	Mixing / Message Processing	43
6.4.1	Tagging Attack Prevention / Verification	43
6.4.2	Replay Attack Prevention	43
6.4.3	Process Intermediate Mix Messages	44
6.4.4	Process Final Mix Messages	44
6.4.5	Mixing Algorithm	45
6.5	Transport Protocol	45

6.6	Dummy Policy	45
7	Prototype Implementation	46
7.1	Feature Subset	46
7.2	Used Technologies	47
7.3	Functional Principle	49
7.3.1	Message Creation	49
7.3.2	Mix Message Processing	51
7.4	Implementation Specials	53
7.4.1	Key Retrieval	53
7.4.2	Prefix Inner Padding	53
7.4.3	Change OpenPGP packet lengths	53
8	Conclusion	55
	List of Figures	58
	List of Tables	59
	Bibliography	60

1 Introduction

The security of email communication and secure messaging in general is a long lasting and still ongoing topic. And becomes even more popular the more information about mass surveillance becomes public.

Basically, messaging protocols can be divided into synchronous and asynchronous delivering strategy. Synchronous chat protocols are designed to provide real-time communication where two or more participants are able to communicate without a considerable delay. In contrast, communicating via asynchronous messaging protocols is more like sending letters. Messages can be delayed from a few seconds up to days. The latter often uses store and forward techniques to deliver messages via one or multiple hops. The oldest and still used protocol of this kind is SMTP [RFC-5321]. But plain emails are highly insecure and non-anonymous and thus can be compared to postcards. Everyone who has access to an email (e.g. Internet and email providers forwarding an email, private attackers or secret agencies listening on connection) can read the whole content as well as the entire metadata including sender and receiver.

Indeed, there are techniques available like transport and end-to-end encryption. While transport encryption is theoretically a good thing it doesn't work reliably in practice for email server-to-server communication¹. Big email providers are fortunately working on more reliable transport encryption using different techniques such as whitelisting, DANE [RFC-6698] or the new proposal of strict transport security (STS) for SMTP [RFC-draft: SMTP STS]. Some providers can already shine with relatively good encrypted communication rates². However, just relying on transport encryption presumes trustworthy service providers involved in the communication path. And even end-to-end encryption solutions for email like OpenPGP [RFC-4880] or S/MIME [RFC-5751] can't protect the metadata of a communication. It's still possible to find out the subject, the sender and receiver and the time of an email conversation. And due to the fact that each

¹Nearly all SMTP servers are configured to offer TLS but don't requires it. This behavior is established in [RFC-3207].

²<https://www.google.com/transparencyreport/saferemail/>, <https://transparency.twitter.com/email-privacy>

server usually appends information to the email header, one can determine the whole path the email took from the sending client to the receiving server.

Therefore, an eavesdropper just needs access to the data of one involved party or the connection between two servers. The first way is given to law enforcement and secret services by law in most countries. Listening on connections is also done by nations secret services and even private hackers are able to do this with a man-in-the-middle attack.

While there are efforts to design and standardize new protocols with built-in security and privacy features to replace the old email protocol, there are also existing protocols to provide privacy features on top of email. But most of these solutions have a high adoption effort or are outdated already. This work is about a secure and privacy preserving asynchronous messaging solution, based on existing email infrastructure and as much established technologies as possible. The main goal is to develop a solution that is easy to integrate into existing infrastructure with low adoption effort.

The requirements for the aimed solution are defined in Chapter 2. After explaining relevant technologies in Chapter 3, existing protocols and projects are described in the Related Work Chapter and will be aligned to the defined requirements. Chapter 5 names facts to respect in order to define a secure and anonymous messaging protocol and discusses design decisions. The centerpiece of this work is the actual protocol specification in Chapter 6. The implementation of a prototype of the specified protocol is explained in Chapter 7. Finally an evaluation of the protocol specification and prototype followed by an outlook to future work is given in the Conclusion.

2 Requirements and Objective

The objective of this work is to develop an on-top solution that uses most of the existing infrastructure and standardized protocols to achieve secure and anonymous email communication. The main goal is that no one except the conversation participants should be able to get full knowledge of the metadata or the content of a message. The concrete threat model is defined in the following section.

The requirements in this chapter are defined for an easy to adopt, secure and privacy preserving asynchronous messaging protocol. They are categorized in dependence to the problem areas and properties described in [Ung+15, Section III.A. and C.] to conversation security, transport privacy, usability and adoption properties. Everything not named in the definition, like the trust establishment problem, is not an explicit part of the requirements and the aimed solution will either inherit its characteristics implicit from the underlying protocols as explained later in the Protocol Specification or simply not focus on it.

The baseline should be a working prototype with good adoption properties and basic security and privacy features (see explanation for the global adversary resistance property).

2.1 Threat Model

Before defining security and privacy requirements it is necessary to name assumed threats in order to aim them specifically.

Local Adversary

An adversary who controls local networks/small network parts, single network nodes or servers (e.g. private attacker).

Global Adversary

An adversary “controlling large segments of the Internet” [Ung+15] (e.g. government surveillance and secret agencies).

Service Provider

Providers have access to data and networks of the services they provide (e.g. email provider, Internet provider).

2.2 Security and Privacy

2.2.1 Conversation Security

Confidentiality, Integrity and Authenticity (CIA)

Confidentiality, integrity and authenticity of exchanged content must be ensured from end to end, i.e. from and to the systems controlled and trusted by the users sending and receiving a message.

Anonymity Preserving

“Any anonymity features provided by the underlying transport privacy architecture are not undermined (e.g., if the transport privacy system provides anonymity, the conversation security level does not deanonymize users by linking key identifiers).” [Ung+15, Section V.A.]

2.2.2 Transport Privacy

Participation Anonymity & Global Adversary Resistance

Nobody except the conversation participants should know that they exchanged messages. Even the participant’s providers should know only one participant of the conversation at most, i.e. that the sender sends or the receiver receives a message. Reaching full global adversary resistance is not trivial and depends also on how heavy the system is used (more messages cause a greater anonymity set) and the effort an attacker could bring up. Therefore, this property is intended but within the limited scope of this work likely not completely satisfiable.

Unlinkability

It should not be possible for anyone except the conversation participants to link multiple messages together and determine that they belong to the same conversation.

Sender Anonymity

Optionally it should be possible for the sender to stay anonymous even towards the recipient. If the senders identity therefor is hidden from the conversation, the

transport protocol must not reveal it.

2.3 Usability

Keep Email Properties

Don't lower basic properties of the the email protocol. Namely asynchronicity, probability of message drops and average message delays.

Easy Initialization

“The user does not need to perform any significant tasks before starting to communicate.” [Ung+15, Section VI.B.] The effort of sending secure and anonymous messages with the solution should not be significantly higher than sending conventional emails.

2.4 Adoption

Compatibility to existing Infrastructure

The solution should be easy to integrate into existing infrastructure and should not require significant additional resources.

No additional Service

No further service infrastructure (e.g. dedicated messaging or directory servers) than already existing and deployed ones are required.

Scalable

“The amount of resources required to maintain system availability scales linearly with the number of users.” [Ung+15, Section VI.C.]

3 Technologies

3.1 Email

Email, or electronic mail, refers to a bunch of protocol and format specifications and software. All components together building up an old and still well used asynchronous messaging system. The first proposal was written in 1973 [RFC-524]. Figure 3.1 shows the overall design for delivering messages by the store-and-forward concept within the email system.

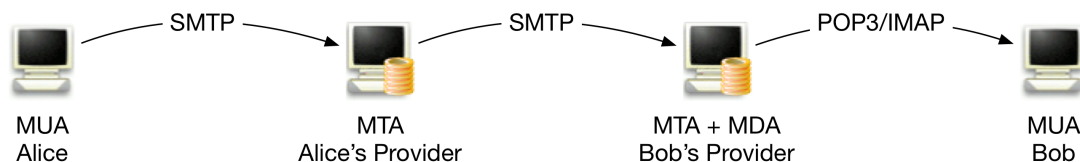


Figure 3.1: Overall Email Design
derived from images at torproject.org by The Tor Project, Inc.,
licensed under CC BY 3.0 US

The different entities showed in Figure 3.1 are:

Mail User Agent (MUA) Usually an email client running on an end user device. E.g. Outlook, Thunderbird, webmail

Mail Transfer Agent (MTA) Server software that stores and forwards emails sent by authenticated users or received for local users. E.g. Postfix, Exim, sendmail

Mail Delivery Agent (MDA) Server software that delivers emails into local mailboxes. Also called Local Delivery Agent (LDA). Often provides services to MUAs to fetch their emails (POP3/IMAP). Usually running on the same server as a MTA. E.g. Dovecot, maildrop

3.1.1 Internet Message Format

The Internet Message Format (IMF) is “a syntax for text messages that are sent between computer users, within the framework of ‘electronic mail’ messages” [RFC-5322, Abstract].

Some parts of the specification are of special interest for this work:

Allowed characters

Only “characters with values in the range of 1 through 127 and interpreted as US-ASCII [ANSI.X3-4.1986] characters” [RFC-5322, Section 2.1] are allowed.

Line length limitation

A line must not exceed a length of 998 characters. A line length of 78 characters is recommended.

Overall message syntax

“Messages are divided into lines of characters. A line is a series of characters that is delimited with the two characters carriage-return and line-feed.” [RFC-5322, Section 2.1] “A message consists of header fields, optionally followed by a message body.” [RFC-5322, Section 3.5]

Header fields

“Header fields are lines beginning with a field name, followed by a colon (‘:’), followed by a field body” [RFC-5322, Section 2.2]. The following header fields are of special interest for this work:

from (required) “specifies the author(s) of the message” [RFC-5322, Section 3.6.2]

to indicates the recipients of the message

message-id a unique identifier for the message

subject human readable “short string identifying the topic of the message” [RFC-5322, Section 3.6.5]

orig-date (required) date and time when the message was complete to send by the creator

3.1.2 Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) [RFC-5321] is a text based protocol generally used to transfer emails in the Internet Message Format. It is used by the sending MUA

to submit a mail to a MTA and for MTA to MTA communication. SMTP by itself does not provide encryption.

The main commands a SMTP client sends to a server in order to deliver a message are:

HELO identifies the connecting client

MAIL FROM announces who originally sends the following mail

RCPT TO announces who should receive the following mail

DATA transmits the actual mail

3.1.3 Transport Encryption

[RFC-3207] defines the STARTTLS extension for SMTP, the “SMTP Service Extension for Secure SMTP over Transport Layer Security” (TLS in general is defined in [RFC-5246]).

But due to the lack of a reliable way of trust establishment, encryption of server-to-server communication is mostly just opportunistic. Even [RFC-3207] advises to not require the “use of the STARTTLS extension in order to deliver mail” for reasons of interoperability. There are a few technologies available to solve this problem. But they are not very practicable and well used at the moment.

Whitelisting Simply whitelist trusted clients/servers and their certificates or shared secrets; does not scale well

Trust on first use (TOFU) trust the certificate shown by the opponent during its first connection; vulnerable to downgrading attacks, see STS below

DNS-Based Authentication of Named Entities (DANE) [RFC-6698] bind certificates to Full Qualified Domain Names (FQDN) using the Domain Name System Security Extension (DNSSEC); practical fails due to the lack of large DNSSEC deployment

Strict Transport Security (STS) for SMTP [RFC-draft: SMTP STS] prevent downgrading attacks after first connection by specifying parameters that must be respected in future connections

Transport encryption does not prevent providers to read the content of an email. For that, end-to-end encryption is needed.

3.1.4 End-to-End Encryption

End-to-end encryption means that the message is encrypted with a key that is only known to the sender and receiver. That means no other entity is able to read the content.

For email there are two standardized technologies to perform end-to-end encryption.

OpenPGP

OpenPGP is an open standard [RFC-4880] which has evolved from Pretty Good Privacy (PGP) created by Phil Zimmermann in 1991. It uses a combination of symmetric and asymmetric cryptography to encrypt data (not only emails) to a public key (owned by the receiver in case of email). Signing, integrity protection and data compression are also parts of the feature set.

An OpenPGP key consists of a public/private key pair, a set of user IDs (name, address, comment) and a set of sub-keys. The trust establishment works decentralized. Each OpenPGP private key can be used to sign an user ID attached to an OpenPGP key. E.g. if Alice trusts Bob's key and he signed Carols key, Alice can also trust Carol. This concept is also known as web of trust.

The OpenPGP data format consists of packets for different functionalities combined together to satisfy a use case. Each packet has a packet header (containing the type (tag) and length among other information) and body. Figure 3.2 shows the usual structure of an OpenPGP message encrypted to a public key.

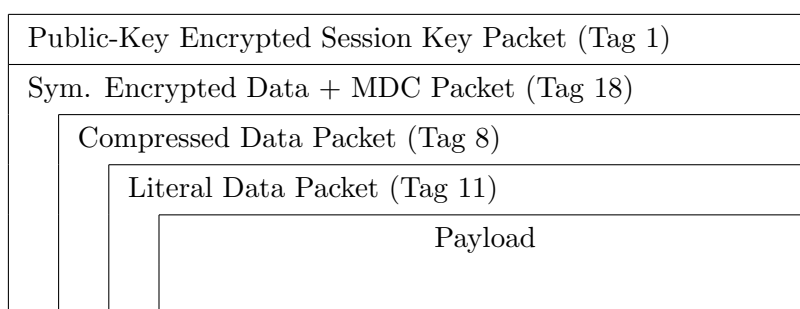


Figure 3.2: OpenPGP Message Format Example

This list names packet types that are of special interest for this work.

Compressed Data Packet (Tag 8) containing a compressed Literal Data Packet

Literal Data Packet (Tag 11) containing the actual payload (text or binary data)

Public-Key Encrypted Session Key Packet (Tag 1) containing a session key encrypted with the public-key of the receiver

Symmetrically Encrypted Data Packet (Tag 9) containing a compressed or Literal Data Packet encrypted symmetrically with the session key contained in the Public-Key Encrypted Session Key Packet

Symmetrically Encrypted and Integrity Protected Data Packet (Tag 18) same as the Symmetrically Encrypted Data Packet appended by a Modification Detection Code (MDC) for integrity checking

OpenPGP currently does not provide perfect forward secrecy. But an extension for this is under development [RFC-draft: PGP PFS].

The most popular implementation of OpenPGP is the free and open-source GNU Privy Guard (GnuPG)¹.

S/MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME) is an open standard [RFC-5751] to encrypt and sign MIME encoded messages using public-key cryptography. In contrast to OpenPGP, S/MIME uses a centralized public key infrastructure (PKI) for trust establishment. The message and key format is not compatible to OpenPGP.

S/MIME is mentioned just for completeness. It is not used further in this work because of the need of a PKI and the lack of well used public key servers to discover keys. Details about why OpenPGP is preferred over S/MIME are described in Chapter 6.

3.2 Mix Networks and Onion Routing

Mix networks and onion routing are both techniques to anonymize the sender and receiver of network traffic. They work in parts similar and are often mixed up even in professional literature [Syv13].

Mix networks were first introduced in 1981 by David Chaum [Cha81]. Mix network messages are routed via multiple nodes (mixes). At these nodes messages are filtered, gathered, re-encoded, delayed and shuffled to blur the trace of a message. The aim is to

¹<https://www.gnupg.org/>

keep the sender and receiver of a message anonymous even against a global passive and active adversary.

Due to the message delay caused by the nature of the mixing process mix networks are basically suited for connectionless high-latency data transmission in just one direction.

The re-encoding is mostly done by a layered encryption of a message to the keys owned by the mixes the message traverses. Each mix then pulls off one encryption layer. This is also a similarity with the onion routing technique.

Onion routing in contrast is designed to provide bidirectional low-latency connections. In general it uses no mixing process. Messages are just re-encoded at each node before being passed on. The security is gained from choosing unpredictable paths that are hard to observe for an adversary. This also means that onion routing networks basically just provide security against a local adversary. Tor is an example for an onion routing network [DMS04].

4 Related Work

4.1 Remailer

Remailers modify and forward emails to achieve anonymity and/or confidentiality for either the sender or receiver or even both. There are different types of remailers categorized according to the level of resistance against de-anonymization by a global adversary.

Type 0 pseudonymous (abbreviated nym) remailer

A remailer mainly designed to replace the real sender address with a pseudonym and allows the receiver to reply to the original sender.

Type I Cypherpunk remailer

A Cypherpunk remailer removes all sender related information and therefore does not support replies. In addition, it is possible to route a message over multiple remailers and encrypt the message between remailers.

Type II Mixmaster remailer

A Mixmaster remailer encrypts and splits a message into multiple parts of equal size and sends them over different paths. Replies are not part of the Mixmaster design.

Type III Mixminion remailer

A Mixminion remailer works like a Mixmaster except the following details: it doesn't use the SMTP protocol within the mix-net, a directory service is used to discover mixes, implements a technique to anonymously reply to messages and some more features.

Of course there are some more remailer implementations like babel [GT96] and reliable/jbn2 [Nym99] (type I/type II hybrid). To give an overview and describe how remailers work it will be adequate to just consider the reference implementations. The other ones mainly work very similar and differ only in some details.

4.1.1 Pseudonymous Remailer

A pseudonymous remailer strips off all sender related data from the header of an email and forwards the message with a newly generated sender address. The mapping of this pseudonym address to the original sender address of the email is stored. Replies to the pseudonym address will be forwarded to the real address by the nym server. A nym remailer can deal with both, encrypted and unencrypted messages. It is also possible to create a so called reply block at a nym server containing a PGP public key of the original sender. The nym server can then encrypt a reply message with the public key before forwarding it to the deposited address.

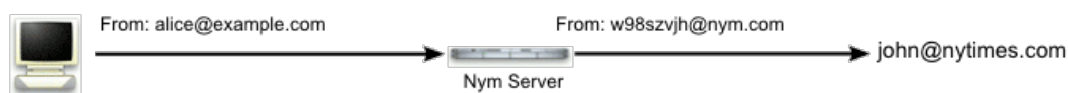


Figure 4.1: Pseudonymous Remailer Scheme
derivated from crypto.is by Tom Ritter,
licensed under CC BY 3.0 US

4.1.2 Cypherpunk Remailer

A Cypherpunk remailer, in contrast to a pseudonymous remailer, anonymizes emails. It removes all sender related data from the header and inserts its own address as the sender. Therefore, it's not possible for the recipient to reply to the original sender. Encryption of a message with the public key of the remailer is supported as well as sending a message via a chain of different Cypherpunk remailers. When combining these two functionalities a message will be encrypted in layers to each remailer's public key in reverse order of its position in the chain. A remailer decrypts one layer on receiving such an email and forwards it to the next remailer until it reaches the final destination. If replies are needed a Cypherpunk remailer must be combined with a nym server.

The protocol is vulnerable to some known attacks [Cot98].

- Tracking messages by the time and order they enter and leave the remailer due to standard Cypherpunk does not specify reordering nor a message pool.
- Tracking messages by their size because the lack of a uniform size (padding and splitting).
- Replay attacks

4.1.3 Mixmaster

The Mixmaster [Moe+04; DSD04] remailer fixes some flaws of the Cypherpunk remailer. Bigger messages are split into multiple parts of the same fixed size, while small messages are padded to prevent message tracking by size. The encryption of the parts is mandatory and they will be routed via different paths of multiple Mixmaster remailers. The last remailer of all paths must be the same so the message can be reassembled correctly.

Mixmaster packets have a dedicated format with a header and body section. The header section contains subsections for each remailer in the path with information about the packet including routing information. A header subsection is encrypted to the public key of the corresponding mix with 1024 bit RSA. The body is encrypted symmetrically with Triple-DES to a session key also contained in the subsection of the header. The packets are transferred as the body of an Internet Message via SMTP.

As like Cypherpunk remailers, Mixmaster remailers don't natively support replies. To prevent replay attacks, remailers store message IDs of processed messages for several days and discard incoming messages that were already processed or which are older than the cache validity.[Moe+04]

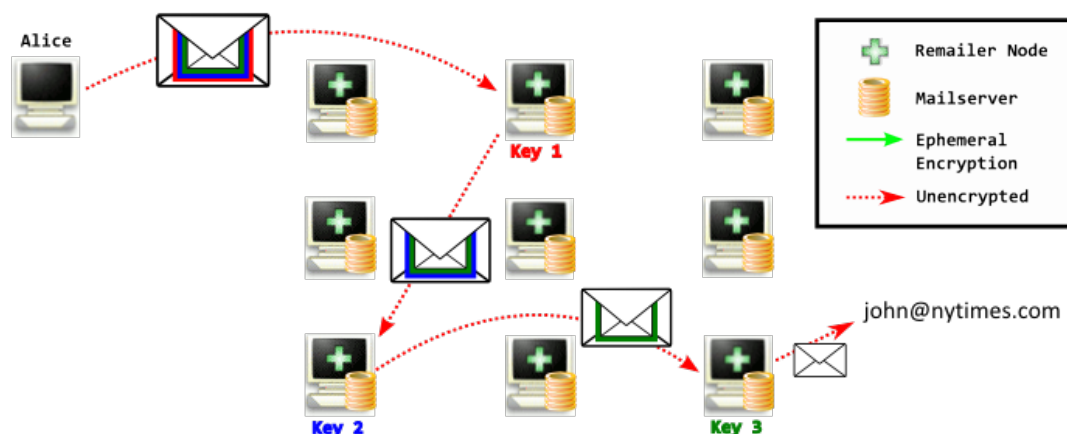


Figure 4.2: Mixmaster Remailer Scheme

original from crypto.is by Tom Ritter,
licensed under CC BY 3.0 US

The Mixmaster specification does not define an own directory service. Instead mixes can just be found via manual maintained lists on the Internet. The availability of a mix can be checked with “Pinger” programs¹. They send status request messages to mixes and interpret the response.

¹e.g. Echolot <https://www.palfrader.org/code/echolot/>

The protocol is vulnerable to some known attacks.

- Flooding attack (see 5.1 Known Attacks to Mix Networks)
- Tracking messages by timestamps [DDM03, Section 5.4]

4.1.4 Mixminion

Mixminion [DDM03] again fixes some flaws of its predecessor. First, Mixminion uses so called single-use reply blocks (SURB) to provide direct or anonymous (recipient anonymity) replies. A message traverses some mixes selected by the sender (first leg) until it reaches a crossover point, then again it traverses some further mixes (second leg) optionally selected by the receiver, communicated through a SURB with a previous message. In other words, a SURB is attached to a message and contains information about the mixes that must be used by the receiver for the second leg to reply to the message.

Second, transport encryption is mandatory for Mixminion communication. But since it uses “TLS over TCP for link encryption between remailers” with forward secrecy instead of SMTP² it is not compatible to Mixmaster and Cypherpunk remailers.

Third, replay attacks are prevented by caching message IDs and periodical key changes. Message IDs of messages encrypted to an expired key are deleted from the cache after that key expires.

Fourth, the protocol design integrates directory servers. These servers provide clients with current information about the Mixminion network including active Mixminion servers, their keys and statistics.

Fifth, to react against exit policy abuse there is a technique defined in order to allow a user to opt-out from a Mixminion remailer, disallowing it to deliver further messages to this user.

Sixth, a mix-to-mix dummy traffic policy is specified. “Each time the mix fires, it also sends out a number of dummies” [DDM03, Section 8.2]. This makes it harder for an adversary to track real messages.

Some theoretical attacks and open problems are discussed in [DDM03, Section 9 and 10] and require further research. But currently no effective attacks are known.

²Mail servers don’t had this feature at the time the Mixminion design paper was written in 2003, e.g. postfix added support for forward secrecy in 2007 (see http://www.postfix.org/FORWARD_SECRECY_README.html, <http://ftp.uma.es/mirror/postfix/src/>).

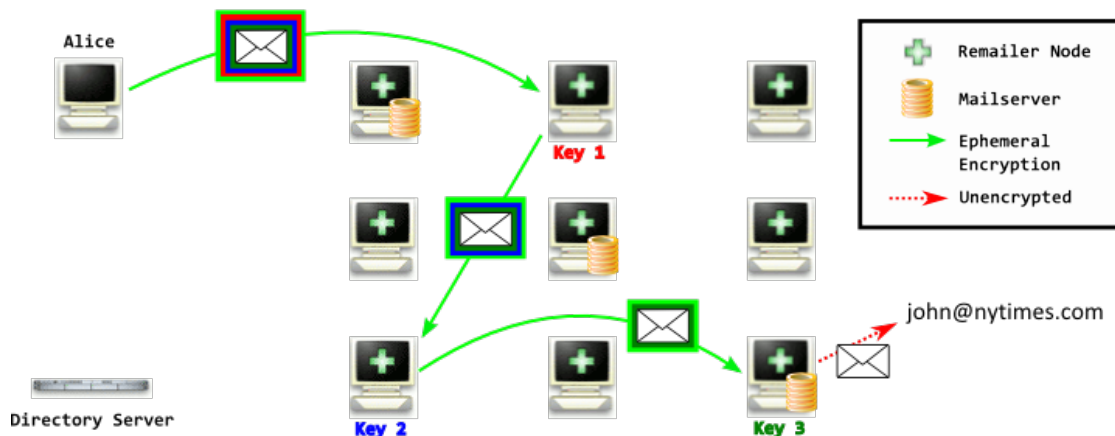


Figure 4.3: Mixminion Remailer Scheme
original from crypto.is by Tom Ritter,
 licensed under CC BY 3.0 US

4.2 Dark Mail

Phil Zimmermann (creator of PGP) and Ladar Levison (Lavabit³ founder), amongst others, are working on an ambitious project called Dark Mail⁴. The object of this project is to define a specification called Dark Internet Mail Environment (DIME). A protocol set to replace current email protocols like SMPT, POP3 and IMAP with built-in end-to-end encryption [Lev+15].

The focus is not explicit on anonymity but the protocol will also approach “a dramatic reduction in the leakage of metadata to processing agents encountered along the delivery path” [Lev+15, Part 1: Abstract].

4.3 Bitmessage

“We propose a system that allows users to securely send and receive messages, and subscribe to broadcast messages, using a trustless decentralized peer-to-peer protocol.” [War12]

A Bitmessage address is a hash of a users public key, so it is directly associated to the users key (in contrast to email addresses and OpenPGP or S/MIME keys). But this makes Bitmessage addresses way more complex for humans to read and share.

³<http://lavabit.com/>

⁴<https://darkmail.info/>

In order to send a message it must be encrypted to the receiver's public key and send as a broadcast message to the peer-to-peer network. The sender must also compute a proof-of-work (partial hash collision) for each message it wants to send to prevent flooding the network and spam messages. The receiver gets all messages addressed to all users on the network and tries to decrypt it with its private key. If the decryption of a message is successful, the message was addressed to this user.

Message are stored for two days by the network in the case that a receiving client is offline. If a client is offline for more than two days a message sent to it during this time must be resent by the sender.

4.4 OnionMail

“OnionMail is an anonymous and encrypted mail server made to run on the TOR network without losing the ability to communicate with the Internet.”⁵

OnionMail servers are communicating to each other via SMTP only with *.onion* addresses through the TOR network. But via exit/enter servers it is also possible to exchange emails with servers outside the TOR network using ‘normal’ domain names. The connections are protected through Tor against local adversaries as far as only OnionMail servers are involved. But also an OnionMail server can read the complete metadata of an email. And in addition when a message is send to or from an usual email server, the same privacy problems exist as with plain SMTP servers.

4.5 Reasons for a new Protocol

None of the existing solutions are or probably will be widely deployed in the near future. Remailer, especially Mixmaster and Mixminion, are a good base but since the specifications are quit old the used cryptographic algorithms (1024-Bit RSA, MD5, SHA-1, 3DES) are already outdated [Moe+04; DDM07]. Indeed it is time to replace SMTP by a protocol with built-in security and privacy features. But it will take a long time before such a protocol becomes an open standard, gets implemented by different makers and will be widely deployed and adopted.

Table 4.1 shows how the described existing solutions fulfill the defined requirements.

⁵<http://en.onionmail.info/what.html>

It points out that no named solution fits all requirements defined in Chapter 2.

Solution	Security		Privacy				Usability		Adoption		
	<i>CIA</i>	<i>Anonymity Preserving</i>	<i>Participation Anonymity</i>	<i>Global Adversary Resistance</i>	<i>Unlinkability</i>	<i>Sender Anonymity</i>	<i>Keep Email Properties</i>	<i>Easy Initialization</i>	<i>Compatibility to existing Infrastructure</i>	<i>No Additional Service</i>	<i>Scalable</i>
SMTP	-	-	-	-	-	-	●	●	●	●	●
OpenPGP, S/MIME	●	○ ¹	-	-	-	-	●	●	●	●	●
Mixmaster	●	●	●	○	●	●	●	●	●	-	●
Mixminion	●	●	●	○	●	●	●	●	○	-	●
DIME	●	-	-	-	-	-	●	●	○	-	●
Bitmessage	●	●	●	●	●	○	○	●	-	○	○
OnionMail	-	-	○	-	-	○	●	●	●	-	●

● = provides property; ○ = partially provides property; - = does not provide property

¹ [Ung+15] say yes but the “Public-Key Encrypted Session Key Packet” contains “An eight-octet number that gives the Key ID of the public key to which the session key is encrypted.” [RFC-4880, Section 5.1] Public key servers seems to disallow search on key IDs but this is no guarantee for anonymity. GnuPG has an option to hide the recipient’s key ID but this isn’t conform to [RFC-4880].

Table 4.1: Requirement Fulfillment of Existing Solutions

5 Design Considerations

This chapter discusses design decisions for a secure and anonymous email and mix network based messaging protocol (remailer). A mix network as the base was chosen because it works well with the store-and-forward concept of SMTP and provides a better privacy level than onion routing technologies.

5.1 Known Attacks to Mix Networks

To prevent already known attacks they need to be taken into account in the design considerations. Attacks can generally be divided into passive (eavesdrop) and active (manipulate data) ones.

5.1.1 Passive Attacks

Correlation by Content

Tracking messages by content should not be possible at all in a mix network. This can be achieved by re-encoding the content through en- or decryption.

Correlation by Size

When messages are not padded to a uniform size it would be possible to track them by their size.

Flow-Correlation Attacks / Statistical Analysis

Assuming a global passive adversary, it could gather information about packet flows by means of correlating and statistical analysis of traffic between the mixes [Zhu+05].

5.1.2 Active Attacks

Replay Attack

Inject copies of the same message again is called replay attack. Deterministic

systems produce same output for same input. Therefore, the attack can be used to map an input to an output message to track it.

Tagging Attack

For a tagging attack a packet is marked (tagged) by an adversary in a way that an honest system will not reject it but the packet is still recognizable by the adversary after it was processed by the system.

Blending Attack / n-1 Attack

If “an attacker targeting a specific message going into a mix can manipulate the batch of messages entering that mix so the only message unknown to him in the batch is the target message” [SDS02], this is called blending attack. “This manipulation may involve delaying or dropping most or all other incoming messages (a trickle attack), or flooding the batch with attacker messages (a flooding attack).” [SDS02]

Partition Attack

If the anonymity set is too large for direct analysis, the adversary could try to partition the anonymity set into smaller subgroups. For example this can be done by manipulating the knowledge a client has about the network. If a client does only know about some few mixes instead of all mixes in the network, an adversary can easier predict or detect the path the client selects for a message [DDM03, Section 1].

5.2 Mixing

As the solution should be build on top of the well established simple mail transfer protocol the design is also bound to its requirements and limitations. A mail transfer agent (MTA) needs to know who sends an email (SMTP MAIL FROM command and from email header field) and to whom the message should go (SMTP RCPT TO command and optional to email header field).

To leave a MTA in the dark about the sender and receiver it is necessary to route the email multilayer encrypted via a chain of mixes each stripping off one encryption layer. Each layer has its own header containing the address of the previous and next hop. Of course the first and last mix in the chain knows at least one of the two communication parties.

The detailed procedure would be as follows:

1. The sending party has to choose a random path through multiple mixes for each email.
2. The whole email including headers should then be encrypted with the key of the last mix and a new header with just the last and previous mix address will be prepended.
3. This encryption is done in layers for each mix in path in reverse order. The first/inmost layer will be encrypted with the key of the last relay and the outmost layer will be encrypted with the first relay in path.
4. A mix must strip off one encryption layer and forward the message to the next mix.
5. After the last encryption layer was removed the local delivery agent is able to store the message in the correct local mailbox (assuming the MTA of the receiver also supports the protocol).

To complicate statistical and blending attacks a mix should additionally implement a pool to gather incoming messages and send them out reordered with random delays. The best strategy to keep a good anonymity set even when a mix is under a blending attack seems to be the timed dynamic-pool [SDS02]. The mix possibly sends out messages every t seconds. But only if the pool contains at least a number of minimum pool size plus an additional threshold of messages. In every period, a fraction of the messages in the pool is sent, the rest is kept in the pool. This strategy is also used by Mixmaster [Moe+04, Section 3.1] and Mixminion [DDM03, Section 8.1] and therefore already field-tested. But specify the exact parameters (time period the mix fires, minimum pool size, threshold and fraction of messages send when firing) is a tradeoff between attack resistance and message latency. Parameter values for great anonymity and attack resistance result in heavy message delays in particular when there are few messages on the mix network. Adapt the parameters dynamically according to the message frequency may be an alternative. But before using such a technique further research is required to rule out negative side effects (e.g. encourage trickle attacks). Another approach to flush out real messages from the pool in times of low user activity is to make use of dummy traffic as described in Section 5.8.

Of course, there exists a bunch of other mixing strategies [SDS02], most mentionable the stop-and-go mix, or continuous mix [DP04]. It delays every incoming message a random amount of time independently instead of collecting messages into a common pool. But this design is vulnerable to tickle attacks without further counter-measures

such as dummy traffic.

5.3 Message Size

Tracking messages by size should not be possible either due to uniform or unpredictable size. Due to the fact that each mix needs some kind of routing information that is stripped off when a message is processed by the mix the message size would shrink at each mix if not padded. Using a fixed size has the advantage that messages could not be tracked by its size neither from an eavesdropper nor from a malicious mix.

For the situation when a large message is split into many small ones or many messages should be send for any other reason it should be specified whether to send them through one or multiple paths. A message flood through a single path is easier traceable. Using many different paths increases the risk of selecting a path where all mixes are owned by an adversary. The truth is somewhere in the middle: Select a few paths and send some messages through each. And even better: send the bunch of messages spread over some time instead all at once [DDM03, Section 8.3].

5.4 Replies and Sender Anonymity

Replying to a message is possible as long as the sender does not hide its own address from the primal header. If the sender does so, it will remain anonymous but cannot receive replies.

As it is not a requirement recipient anonymity and anonymous replies must not be supported. If these features are requested regardless the requirements, a technique like the single-use reply blocks defined in [DDM03] must be implemented. Without recipient anonymity it is always the choice of the sender of an initial or reply message to send it anonymized or not through the mix network¹. If someone does not want to receive direct messages, he has to configure his MTA to only accept messages from the mix network. But this will not prevent that such messages are sent and transferred over the network. Again, to solve this issue recipient anonymity and anonymous replies would be needed.

¹This is not more adventurous than the real world: Alice: “Please don’t talk about my problem here.”, Bob: “Why? Your problem. . .”.

5.5 Mix Discovery

When a client wants to send a message it needs a list of available mixes that can be used to select the path. It is desirable that each client has the same view at the current network. Otherwise the anonymity set for the sent message would shrink to the view of the respective client. Therefore, presenting a client a limited or modified set of mixes could be an attack vector (partition attack) [DDM03, Section 6].

The actuality of the list of available mixes is also crucial for the reliability of the selected path. If the list contains unavailable mixes, the message cannot be delivered. Selecting paths on the basis of an additional ranking system (higher ranking for long time available mixes) could improve the reliability. Such a ranking system could also minimize the chance of using malicious mix nodes just added in great numbers (because of their probable low ranking).

To satisfy the requirement of no additional service it is necessary to reuse an existing service as directory service or to use a solution that does not require a centralized service (e.g. with use of distributed hash table (DHT)) to discover available mixes.

Retrieving available mixes must happen in a way that neither the directory service provider nor any other adversary can infer the specific mixes that will be used by the client to send the message. This means querying the directory service must happen securely and anonymously or via some private information retrieval (PIR) functionality. PIR is a technique to retrieve a dataset from an entity in an way that it does not exactly know which one was requested. In its simplest form that can be done by requesting all datasets even if just one is needed.

5.6 Replay Attack Prevention

Incoming messages should be checked in some way and possibly discarded if they were processed earlier or were expired already to prevent replay attacks. This could be done like in the Mixminion protocol by storing hashes of processed messages in combination with using short lifetime sub-keys for the mixes. The cache of message hashes could then be flushed after key expiration. Using sub-keys with short lifetime has also the advantage of improved forward secrecy.

5.7 Tagging Attack Prevention

To prevent tagging attacks it is necessary to detect a tagged message before it reaches a point later in the path where the attacker is able to recognize the tag. Assuming the attacker is not able to recognize a tag in the encrypted part of a message² it is adequate to do an integrity check at each mix, just over the part that can be decrypted to plaintext by the next mix. If the integrity check fails, the message must not be forwarded.

5.8 Dummy Traffic

Dummy traffic, or cover traffic, could be used to increase the anonymity set of messages in times of low user activity. And additionally if using a threshold based or a timed dynamic-pool at the mixes, it could also lower message delays in these times (details described below). In order to become indistinguishable from user generated traffic dummy traffic must have similar properties in terms of message size, passing multiple mixes and frequency. It can be differentiated into messages between servers only and messages involving clients. Of course the latter gets better protection against a global adversary, but is even more complex [BL02] and still requires some more research.

For server-to-server dummy messages there are some practicable dummy policies. One way is to send out one or more dummy messages additionally each time the mix fires at least one message out of the pool [SDS02, Section 5.3] (flush triggered). This makes sure that even if an attacker is able to flood the pool at least one message unknown to the attacker comes out each time. Another policy is to add a random number of messages to the pool before each execution of the mixing algorithm (time triggered). This would have the side effect, as mentioned above, that queued messages become flushed out after some time and also if there are no new incoming messages. And the third way is to add a random number of messages each time a new message comes in (message triggered).

Mixmaster specification defines using time (“one every 9 mixing rounds”) and message triggered (“one dummy for every 32 messages coming in”) dummy message creation [Moe+04, Section 3.2].

²The output of a single decryption operation of double encrypted text is not predictable = random bytes to one who cannot finally decrypt it to plaintext.

5.9 Abuse

As every other technology or tool, anonymized email communication can also be abused. And as it is in the real world as well it cannot be completely prevented. Every known technique to prevent or react on abuse also weakens the anonymity and privacy of the system.

A mix can just transfer mix messages (middleman node) or operate as an exit node which delivers messages locally (private exit node) or through the network (open exit node). Open exit nodes are extra vulnerable to abuse because they have a great range. And other systems may see them as the originator of such messages (the open exit node is the only entity known to others, the sender remains anonymous).

5.9.1 Opt-out Mechanism

The Mixminion design uses an opt-out mechanism – a receiver can request a mix to not send him any further mails [DDM03, Section 5.3]. But even this is a all-or-nothing solution which does not get down to the root of the trouble.

5.9.2 Spam Resistance

Spam and spam prevention is a relevant topic for all kinds of messaging solutions. In general it should not be that attractive to send encrypted spam mails because of the higher cost. But the anonymization could allow spam sending entities to remain in the dark and not to get blacklisted. And if a mix allows to relay anonymized messages (open exit node) it is also possible to run a denial of service attack against this node by provoking that it gets blacklisted.

Since this work is about a solution on top of email to add security and privacy features it must not have necessarily a better spam resistance than its underlying technology.

Implicit Spam Protection with SMTP

SMTP by itself does not provide explicit spam protection. But because spam sending clients often do not meticulously redeem the protocol compliance it is established practice to block such clients. Indicators for an innocent SMTP client are for example:

- authentication (with username/password or certificate)

- FQDN hostname and matching HELO name
- DNS entry (MX or A record) for hostname and HELO name
- reverse DNS entry for client IP (PTR record)
- strict sequence of protocol message exchange
- greylisting (cancel first connection attempt, accept second attempt)

All techniques named above can potentially also be used by the aimed solution as well.

Explicit Spam Protection for Email

Additionally there are some other techniques available to detect spam:

- content based detection
- blacklisting
- Sender Policy Framework (SPF) as sender spoofing protection
- DomainKeys Identified Mail (DKIM) as sender spoofing protection

This techniques can only be used if the message is not end-to-end anonymized and encrypted. They could potentially be used by relaying mixes.

Cost-based Spam Resistance

One property why sending spam is lucrative is because it is very cheap. Increasing the cost (monetary or computation power) of sending a message may lower the spam problem.

Creating a valid message for a mix network (path selection, key retrieval, encryption) is already relative costly compared to a normal email. This may be enough or can be complemented by additional cost increasing techniques.

6 Protocol Specification

This chapter describes the specification of a new protocol for secure and anonymous communication as an on top solution using most of existing infrastructure and technologies. The requirements as defined in Chapter 2 and design considerations from Chapter 5 are considered.

Due to the fact that the solution should be compatible to existing infrastructure and require no additional services, it would be best to integrate the logic of a mix as a module for existing MTA software and wrap the data in a valid Internet message (email) [RFC-5322] and use SMTP for transport. Also an existing encryption system, namely OpenPGP, is utilized. By using existing PGP key servers not only for key distribution but also as discovery service to find other available mixes no further service is needed. The web of trust concept used by OpenPGP provides additional flexibility that could not be reached with a PKI.

6.1 Introduction

In the following the “Mixtasy” protocol and message format (version 1) is specified. Mixtasy belongs to the group of remailer protocols. The specification is based on several existing technologies namely SMTP [RFC-5321], Internet Message Format [RFC-5322] and OpenPGP [RFC-4880].

6.1.1 Terms

Sender The user who is the originator of a message.

Recipient The user to which the message is addressed.

Provider The entity that operates the MTA/MDA for the sender or receiver.

Mix A MTA which is running an implementation of the protocol to process such messages.

Mix Message A Protocol message with a mix message header containing special fields for processing and encrypted payload. Intermediate and final mix messages were differentiated.

Original Message The Internet Message as composed by the sender.

Directory Service The service, here an PGP key server, which provides the clients with a list of available mix nodes.

Mix Address Each mix has an email address in the form of “mixtasy@FQDN”.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC-2119].

6.1.2 General Functions

This protocol extends email conversation by:

- conversation security and integrity
- participant anonymity
- optional sender anonymity
- all OpenPGP functions (digital signatures, encryption, compression, Radix-64 conversion) by using it as an underlying technology

6.1.3 Overall Design

The overall design of the protocol is a mix net using SMTP and Internet Message Format for exchanging protocol messages. While the sender is required to craft a protocol message in a specific way, the receiver just receives an email in the Internet Message Format either in plaintext or preferable encrypted/signed with OpenPGP. Any existing MTA can turn into a mix node by publishing an OpenPGP key on a key server and running some protocol specific logic beside the MTA software.

A message MUST traverse at least four mixes. If a provider of the sender or receiver is also a mix, it MUST be included in the path. Three or less mixes would not be enough to also obscure the sender’s and receiver’s provider. In case when both providers are mixes, by using just three mixes the middleman mix would know which provider has sent and received the message.

Figures 6.1 and 6.2 showing the overall design of the Mixtasy Protocol in the cases the provider of the receiver supports the protocol or not.

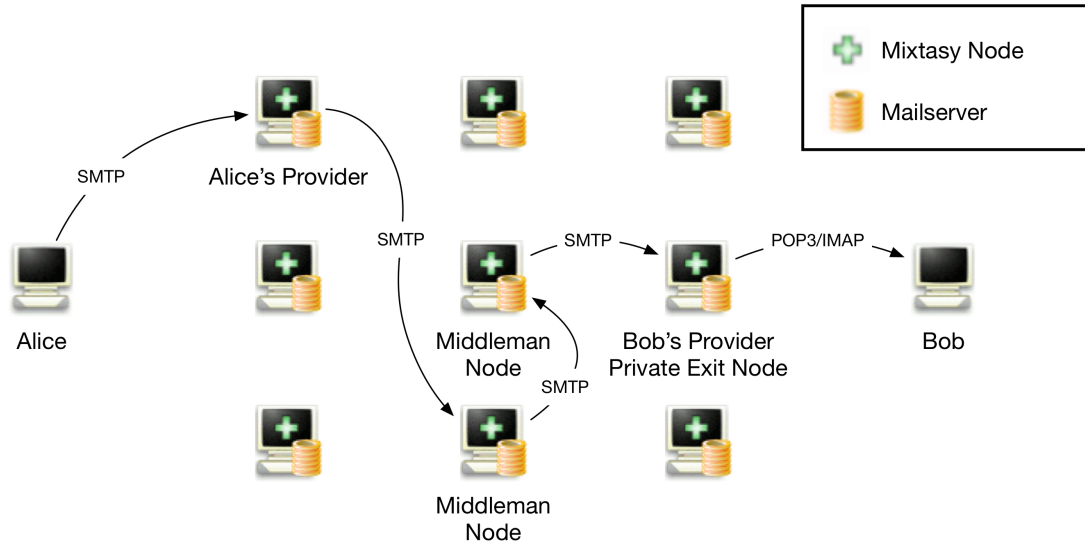


Figure 6.1: Mixtasy Design: Receiver's provider supports the Mixtasy protocol
derived from images at torproject.org by The Tor Project, Inc.,
 licensed under CC BY 3.0 US

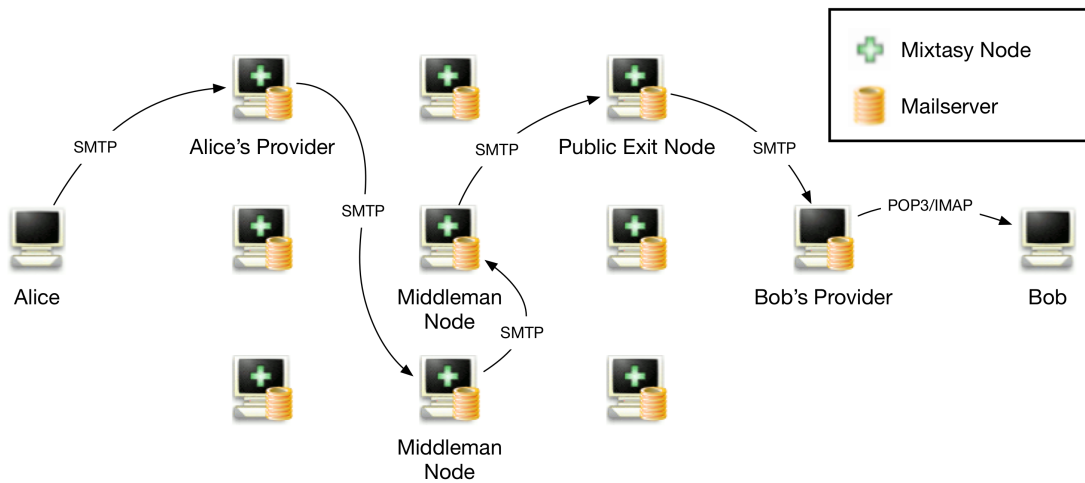


Figure 6.2: Mixtasy Design: Receiver's provider does not support the Mixtasy protocol
derived from images at torproject.org by The Tor Project, Inc.,
 licensed under CC BY 3.0 US

6.2 Message Format

The Mixtasy message format is mainly a layered Internet Message with an OpenPGP encrypted body (the encrypted body contains another Internet Message with an encrypted body and so on). An intermediate mix message consists of a header (retains routing and verification information used by the mix) and an encrypted body containing either another intermediate mix message or a final mix message (which by itself contains the original message).

The most inner message is the original message as composed by the sender. The body of the original message **SHOULD** be encrypted to the receiver and optionally signed by the sender to approach end-to-end conversation security and authenticity.

Because of the uniformed size of Mixtasy messages big original messages have to be split into multi part messages.

Figure 6.3 illustrates the explained message format.

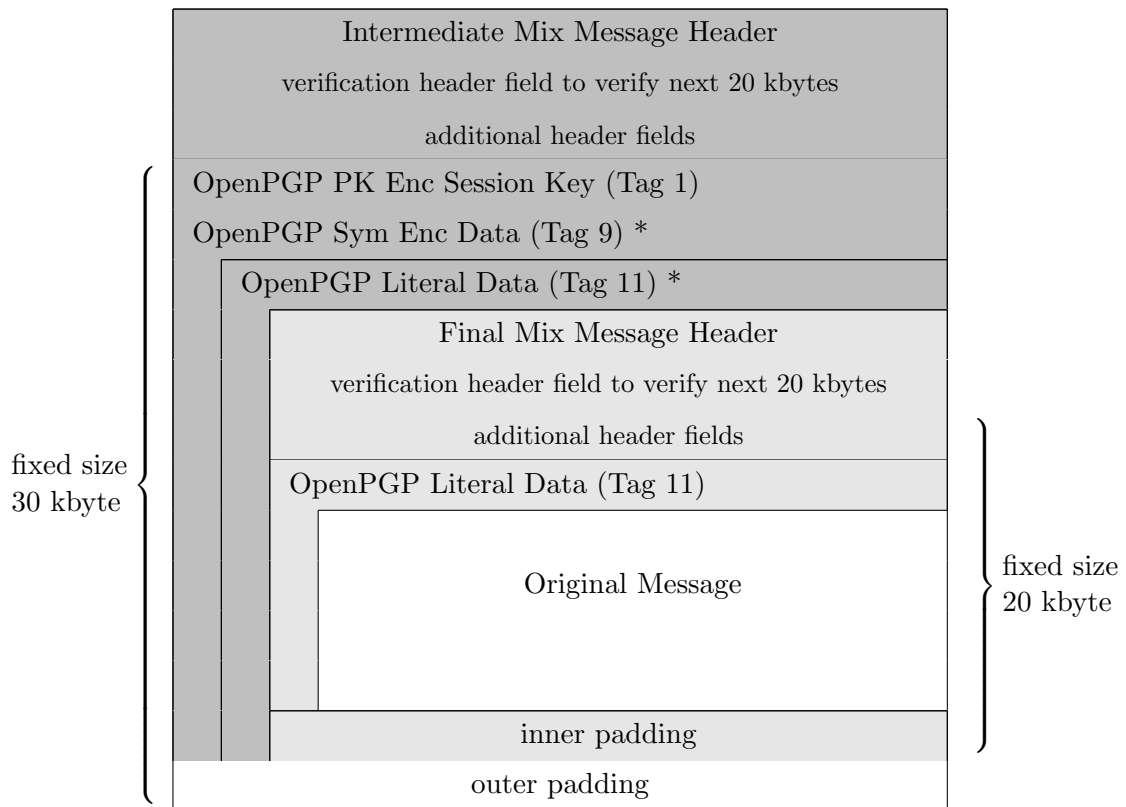


Figure 6.3: Mix Message Format

- * = changed length to 30 kbytes or indeterminate length
- = intermediate mix message (repeated for each mix in path)
- = final mix message

6.2.1 Original Message

The original message SHOULD be [RFC-5322] compliant and additionally MUST have a **To:** header field containing the email address of the receiver.

To achieve end-to-end conversation security and privacy the body of the original message SHOULD be encrypted to the receiver using OpenPGP and if not sent anonymous, also be signed by the sender. If the message is encrypted the header MUST NOT contain any sensitive information about the sender (e.g. original sender address in the **From:** field), context (e.g. message id in the **Reply-To** field) or content (**subject** field) of the message.

6.2.2 Final Mix Message

Final mix messages are only processed locally by the last mix in the path (this does not apply to the original message in the payload). The header of this messages MUST meet the requirements shown in Table 6.1.

The body of a final mix message is an OpenPGP Literal Data Packet containing the original message, or in case of a multi part message just one part of it, possibly followed by padding. In order that the verification hash covers the whole final mix message (see Section 6.4.1 for details) it MUST be exactly 20 kbyte in size during creation, excluding the verification header field. Smaller messages MUST be padded with random bytes. The Literal Data Packet can be separated from the padding by the final mix due to the fact that the length of each OpenPGP packet is stated in its header [RFC-4880, Section 4.2]. If the original message is too big to fit into a single part message, it MUST be split into multiple parts. Each part is wrapped into a Literal Data Packet representing the payload of a multi part message.

The final mix message header also contains information about the type of the payload. The type could be a single part message, a multi part message or a dummy message. While single part messages could be processed directly, multi part messages must be reassembled first. The payload of a single part or reassembled multi part messages MUST be handled like an usual email. Dummy messages get destroyed. Further types could be defined in future.

6.2.3 Intermediate Mix Message

An intermediate mix message SHOULD comply to [RFC-5322] and MUST meet the requirements shown in Table 6.1.

The body of an intermediate mix message MUST be either another intermediate or final mix message encrypted to the mix given in the To: header field using OpenPGP. The encrypted packet MUST NOT contain a modification detection code and the content within MUST NOT be compressed. The length stated in the OpenPGP Symmetrically Encrypted Data Packet MUST be set to 30 kbyte or indeterminate length. This also applies to the OpenPGP Literal Data Packet inside. In combination with added padding this makes it impossible to detect the message length of the contained message.

The body of the outmost intermediate mix message MUST be padded with random bytes to a size of 30 kbyte.

Random padding bytes are indistinguishable from encrypted bytes. The decryption of an encrypted message with trailed random padding will result in the unaltered message but followed by random bytes. In the case of the intermediate mix message the decryption reveals another message with encrypted content at the end (followed by random bytes = decrypted random padding). This causes that the real message length cannot be detected until the final mix message was decrypted.

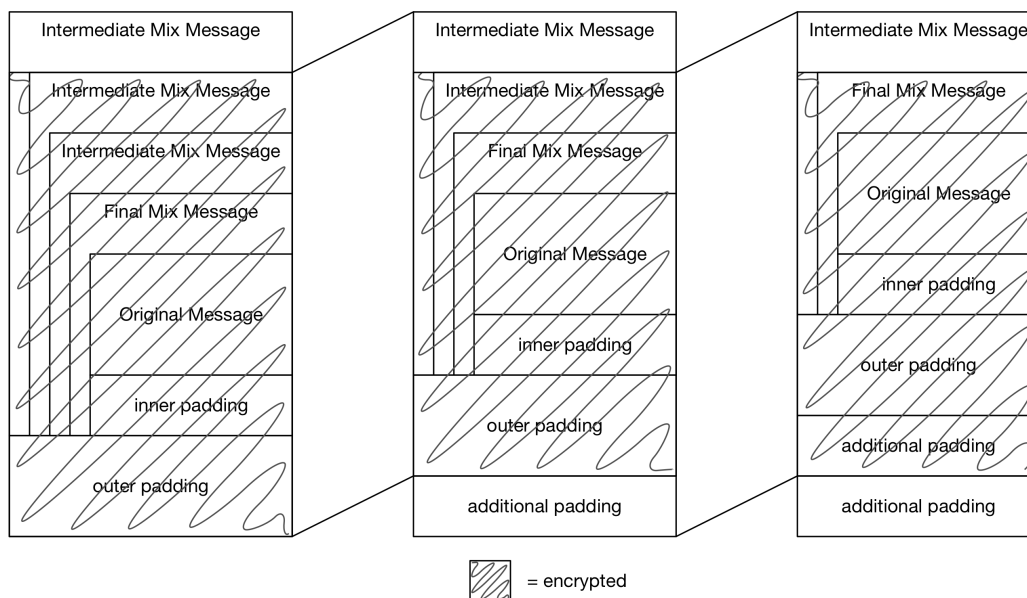


Figure 6.4: Intermediate Mix Message Transition

6.2.4 Header Fields

The overall format of header fields is the same as specified in [RFC-5322]. In supplementation, additional header fields are defined and for some existing header fields other requirements are applied.

Field	Intermediate Mix Message Header	Final Mix Message Header
remailer-type	MUST Mixtasy v1	MUST Mixtasy v1
verification	MUST SHA-1 hash	MUST SHA-1 hash
from	MUST current mix address	-
to	MUST next mix address	-
mixtasy-id	MUST NOT contain this field	MUST unique id
mixtasy-type	MUST NOT contain this field	MUST type
part	MUST NOT contain this field	MUST if type is multipart
other fields	-	-

- = MAY NOT contain this field

Table 6.1: Header Field Requirements

Remailer Type

```
remailer-type = 'Remailer-Type: Mixtasy v1'
```

The remailer-type field states the type and version of the remailer message. It is also used by existing remailer protocols, e.g. Mixmaster. The presence of this field indicates that a message belongs to a remailer protocol. The value indicates more specific that it is a Mixtasy message of protocol version 1.

From Field

```
from = 'From:' mix-address
```

As specified in [RFC-5322] but MUST be the address of the sending mix instead of the author of the message.

To Field

```
to = "To:" mix-address
```

As specified in [RFC-5322] but MUST be the address of the receiving mix instead of

the primary recipient of the message.

Verification Field

```
verification = 'Verification:' SHA-1-hash
```

The verification header field **MUST** be the first one in a mix message header. The verification field **MUST** exactly consist of the name “Verification” (case-sensitive), followed by a colon, followed by a space, followed by the value. An implementation **MUST NOT** do any interpretation or error correction if the field name and name-value-separator is not exactly equivalent. Otherwise this would be a weak point for tagging attacks because the hash did not include the field name.

The value **MUST** be a SHA-1 hash of the next 20 kbyte of the message (including the header and a part of the body) starting at the following line.

Mixtasy-ID Field

```
mixtasy-id = 'Mixtasy-ID:' id
```

The value for the mixtasy-id field provides a unique ID for the original message contained in the final mix message. It **MUST** only occur in a final mix message header. Multi part messages belonging together **MUST** have the same mixtasy-id. The format and constraints for the value are the same as for the message-id field in the Internet Message Format [RFC-5322, Section 3.6.4]. The id **MAY** be the same as the message-id used for the original message.

Mixtasy-Type Field

```
mixtasy-type = 'Mixtasy-Type:' type
```

The mixtasy-type field **MUST** only occur in a final mix message header. It indicates the type of the payload of a final mix message. The value **MUST** be one of the following:

- dummy
- singlepart
- multipart

Part Field

```
part = 'Part:' number/total
```

The part field MUST only occur in a final mix message with “multipart” as the value for the mixtasy-type field. The value MUST then be two numbers separated by a slash character (/). The first number gives the order of this message in multiple messages belonging together. The second number gives the total number of messages belonging together.

This information is used by the final mix to reassemble multi part messages.

6.3 Key Management and Directory Service

6.3.1 Key Generation

A mix needs some asymmetric key material, so that a sending client can encrypt mix messages to the mix. Specifically each mix MUST own a long-term OpenPGP key and multiple short-term encrypt only sub-keys. Only sub-keys MUST be used for encryption/decryption operations. While the long-term key is used for trust establishment, using short-term sub-keys for encryption contributes a certain level of forward secrecy. The short-term validity of the sub-keys is also important for the replay attack mechanism, see Section 6.4.2.

OpenPGP keys have no valid-from property but an expiration date. A key is always valid from the time of creation until it expires. An available mix MUST publish a valid sub-key. The mix MUST create and publish a new valid sub-key when the half of validity period of the previous key is over and it wants to continue to provide its service. As a result an operating mix has published two valid sub-keys at each point in time. The period of validity of the sub-keys SHOULD be two days. Figure 6.5 illustrates the key format and the sub-key rotation.

The OpenPGP key MUST have only one user id containing the address of the mix the key belongs to. The name can be chosen freely. The comment field SHOULD be used to indicate the features the mix supports. Features are stated with a key identifier followed by a colon, followed by a comma separated list of values. No whitespace characters are allowed in a feature statement. Multiple feature statements MUST be space separated. Currently only the “mixtasy-mode” feature is defined which indicates the operation modes of the mix. Valid values for the Mixtasy operation mode feature are `middleman`,

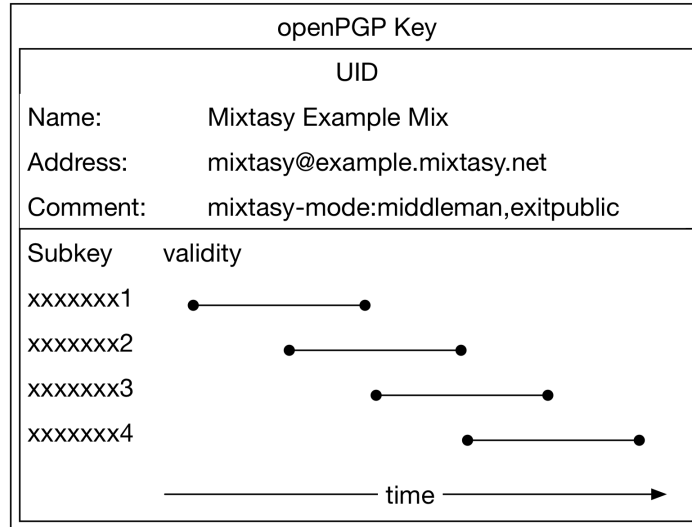


Figure 6.5: Mixtasy’s OpenPGP Key Format

exitprivate, exitpublic. See Section 6.4 for details on operation modes.

6.3.2 Key Distribution and Discovery

The keys SHOULD be distributed over public PGP key servers like <https://pgp.mit.edu/>. Most key servers synchronize known keys, so the keys get distributed over multiple servers.

To discover available mixes one can search public PGP key servers for the Mixtasy mix address scheme. Searching and retrieving keys MUST happen in a way that no information about selected message paths is revealed towards any instance. Either all found keys must be retrieved or the keys must be retrieved in an anonymized way¹. Before using encountered keys it MUST be checked, if they comply to the key format described above.

6.3.3 Ranking

A ranking system for mixes and their keys SHOULD be established. This SHOULD be done by neutral instances by signing mix keys with their own OpenPGP key. Different

¹e.g. GnuPG supports anonymous key retrieval via Tor since version 2.1.10 (see <https://lists.gnupg.org/pipermail/gnupg-announce/2015q4/000381.html>).

ranking levels MAY be described by different keys used for signing or distinct certification levels. The ranking could describe a level of reliability and trust.

6.4 Mixing / Message Processing

A mix can support any combination of three different modes.

middleman node (`middleman`)

The mix only processes mix messages. Final mix messages are discarded.

public exit node (`exitpublic`)

The mix only processes final mix messages. Mix messages are discarded

private exit node (`exitprivate`)

The mix only processes final mix messages which are addressed to local mailboxes.

Mix messages and final mix messages address to remote recipients are discarded.

When a mix receives a mix message addressed to itself it MUST strip off the header and decrypt the body with one of its valid sub-keys. If the message is not encrypted to one of its valid sub-keys the message MUST be destroyed. After successful decryption it MUST be checked if the structure of the revealed message meets the specification of an intermediate or final mix message. Then the mix proceeds with further steps explained in the following subsections. Finally, before the mix finish the message processing by delivering it to the next hop it SHOULD check and if necessary alter the message to be conform with the Internet Message Format [RFC-5322].

6.4.1 Tagging Attack Prevention / Verification

To prevent tagging attacks a mix computes a SHA-1 hash over the first 20 kbyte of the message beginning at the second line. The first line MUST provide the verification header field which contains the SHA-1 hash for comparison. If the computed hash is not equal to the hash given in the verification header field the message MUST be destroyed.

6.4.2 Replay Attack Prevention

To prevent replay attacks the hash from the verification header field MUST be checked against a local cache of such hashes. If the hash is already in the cache the message MUST be destroyed. Otherwise it gets processed further and the hash MUST be stored

in the cache until the expiration date and time of the sub-key which was used to decrypt the message.

6.4.3 Process Intermediate Mix Messages

If the message is another mix message, random padding **MUST** be added to the end of the OpenPGP Symmetrically Encrypted Data Packet to comply to the uniform size requirement (see Section 6.3). After that, the message is put into the mix message pool (see 6.4.5 Mixing Algorithm).

6.4.4 Process Final Mix Messages

If the message is a final mix message it is required to make some further processing in dependence of the type of the payload declared in the mixtype-type field.

Single Part Message

The payload of a single part message **MUST** be an OpenPGP Literal Data Packet followed by padding. The length of the Literal Data Packet is stated in its packet header [RFC-4880, Section 4.2]. Data followed by the packet can be eliminated as it is just random padding. The extracted content of the Literal Data Packet **MUST** be extracted and treated as a usual Internet Message.

Multi Part Message

Like the single part message, the payload of a multi part message **MUST** be an OpenPGP Literal Data Packet followed by padding.

Before processing such messages, the mix **MUST** wait until all multi part messages belonging together arrived. An identical value in the mixtype-id header field indicates related messages. The part field provides the number of messages related and the sorting. Then the contents of the Literal Data Packets of each multi part message **MUST** be extracted and concatenated in the provided order. The resulting original message **MUST** be treated as a usual Internet Message.

Dummy Message

The payload of a dummy message consists of random bytes and MUST be discarded.

6.4.5 Mixing Algorithm

If all previous checks were successful, the message MUST be put into a pool rather than send it out directly. The mix sends out messages from the pool following the timed dynamic-pool scheme.

The mix checks every t seconds if there are at least $n + f_{min}$ messages in the pool, and if so, sends out $(m - f_{min}) * frac$ random messages from the pool, where m is the number of messages in the pool.

6.5 Transport Protocol

For transportation of the protocol messages SMTP, as specified in [RFC-5321], MUST be used. It is highly RECOMMENDED to send and receive messages via TLS with perfect forward secrecy.

Implementations SHOULD be designed as modules/plugins for existing MTA software (e.g. as a filter for postfix [Pos]).

6.6 Dummy Policy

Dummy traffic is used in Mixtasy for two reasons which become relevant in times of low traffic: as a countermeasure for blending attacks and to flush out normal messages from the pool to decrease message delays. Therefore, two different dummy traffic methods are used by Mixtasy:

flush triggered Each time the mix sends out messages from the mix pool a number of dummy messages MUST be send at the same time.

time triggered Before the mixing algorithm gets executed a number of dummy messages MUST be added to the pool.

The payload of a dummy message MUST be random. The header field type of the final mix messages MUST be set to “dummy”. Dummy messages MUST traverse a number of one to four mixes randomly selected for each message.

7 Prototype Implementation

A prototype was implemented to demonstrate the overall practicability of the Mixtasy specification. This means the prototype does not have the aspiration to implement the whole specification but a subset of basic features. The prototype proves that the theoretical approaches are practicable and realizable with low effort. In particular, the prototype concentrates on features that are new in comparison to existing remailer approaches.

The prototypical implementation, its source code and instructions how to use it are available on GitHub: <https://github.com/jojoob/mixtasy>

7.1 Feature Subset

As mentioned before, the prototype implements just a subset of features from the specification. The following features are supported by the implementation.

Message Creation

- message format
- mix discovery and key retrieval via public PGP key servers
- path selection
- single part messages

Mix Message Processing

- strip of an encryption layer
- verification check
- re-padding to fixed message size
- process single part final mix messages

- interface for integration as a simple content filter for Postfix

Not Supported Features

- multi part and dummy messages
- mixing algorithm
- dummy policy
- replay attack prevention

7.2 Used Technologies

The following list gives an overview of software, libraries and programming languages used for the prototype. Each technology is explained in more detail below.

Python <https://www.python.org/>

A high-level interpreted programming language, used for the main application logic.

GnuPG <https://gnupg.org/>

An OpenPGP implementation, used for OpenPGP related operations.

python-gnupg fork by Johannes Burk: <https://github.com/jojoob/python-gnupg/>¹

A Python wrapper for GnuPG.

Bash Shell <https://www.gnu.org/software/bash/>

Used for an adapter script between the python program and Postfix.

Postfix <http://www.postfix.org/>

SMTP server, used to send and receive intermediate mix and original messages

The main application is written in Python and provides a command line interface. Python is a runtime interpreted high-level programming language. Using Python makes it easy and fast to write prototypes and powerful applications among others because the low-level memory management is done by the interpreter, the source code must not be compiled before execution and many built-in high-level methods and additional libraries are available. For some productive systems compiled programs may fit better due to the fact they do not depend on additional installed interpreters. But this argument is not

¹derived from python-gnupg fork by Vinay Sajip: <https://bitbucket.org/vinay.sajip/python-gnupg/>

valid for a prototype which is not meant to be deployed on productive systems.

The most OpenPGP related operations (key search, encryption, decryption, etc.) are done with GnuPG. GnuPG (GPG) is a command line application written in C and, as mentioned earlier, the most popular free and open-source implementation of OpenPGP. With GnuPG Made Easy (GPGME), a software library with basic functionalities of GPG is also provided. The GPG manuals advice to use GPGME in own software rather than to invoke GPG as a subprocess. This is because GPGME provides a stable interface to other programs while the GPG command line interface may change between versions. Unfortunately GPGME does not provide the full range of functions that GPG has. But some of the missing functionalities are required to implement the Mixtasy specification. These are namely:

Store action (--store)

Wraps the data in an OpenPGP Literal Data Packet. Needed to create the final mix message.

Change compression level (--compress-level)

Compression is enabled by default (with level 6). Needed to disable compression (level 0).

Encryption without a modification detection code (MDC) (--disable-mdc)

Symmetrically Encrypted and Integrity Protected Data Packets are used by default for encryption. Needed to just create Symmetrically Encrypted Data Packets.

Search and import keys from key servers (--search-keys)

Needed to discover mixes and retrieve it keys.

This requires to use GPG directly against the commendation. There already exist some Python wrappers for this purpose. But even these modules do not provide all the required functionalities. Therefore, the one which supports the most of the required features was selected (python-gnupg fork by Vinay Sajip) and extended by the missing functions.

The decryption, verification and padding part of processing a message at a mix is also done by the Python program. Receiving and sending intermediate mix and original messages is done by the Postfix server via SMTP. A simple Bash script is used as an interface between the Python program and Postfix.

7.3 Functional Principle

The concrete prototype is a combination of a Python program `mixtasy.py`, a Bash script `mixtasy-filter.sh`, configuration instructions for the Postfix server and a helper script `sendsmtp.sh`.

The `mixtasy.py` provides two main actions via a command line interface. Create a Mixtasy message from an original message provided by the user (`--create`) and process a mix message as it would be done by a mix (`--unpack`). Listing 7.1 shows how to invoke the program on the command line.

```
usage: mixtasy.py [-h] [-m FQDN] [-u] [-f file] [-o file]
```

Mixtasy - an OpenPGP based remailer

optional arguments:

```
-h, --help            show this help message and exit
-c, --create          Create a mix message from a supplied email.
-m FQDN, --first-mix FQDN
                      FQDN of a mix used as the first one in
                      the generated path (if your provider
                      operates a mix, use that one)
-u, --unpack          Decrypt a mix message and verify payload
-f file, --input-file file
                      Read input from file instead of stdin
-o file, --output-file file
                      Write output to file instead of stdout
```

Listing 7.1: `mixtasy.py` usage

7.3.1 Message Creation

The message creation process straight follows the specification. First of all the user must provide an Internet Message with one recipient stated in the to header field, this will be treated as the original message. After parsing the user input all available mix keys are retrieved from the public PGP key server `hkp://pgp.mit.edu`. Then a path of mixes will be selected. If a Mixtasy mix key is found for the FQDN of the recipient it will be used as the last mix. A specific mix can be selected as the first mix in the path with the `--first-mix` option. Afterwards the path will be filled up to four mixes by choosing random keys from all locally available valid Mixtasy mix keys. The same mix will not be

used twice in succession. Next the original message as provided by the user is wrapped into a final mix message and padded. Finally the final mix message will be wrapped in layered intermediate mix messages as specified by the message format for each mix in the path.

The generated message must then be send via SMTP to the first mix. This can be done with the contained helper script `sendsmtp.sh` which sends a message given in the second argument to a recipient given in the first argument via the local running SMTP server. Listing 7.2 shows the sending process of a message.

```
1 $ ./mixtasy.py -m 1.alphatest.mixtasy.net -o mixmessage
2 Type in a message.
3 Finish input with Ctrl+D or by enter a line containing only a . character.
4 To: bob@2.alphatest.mixtasy.net
5 From: alice@1.alphatest.mixtasy.net
6 Subject: Just an email send via Mixtasy
7
8 Hi Bob,
9 this is a secret Message.
10 All the best, Alice
11 .
12 INFO create a message...
13 INFO search key for mixtasy nodes on keyserver 'hkp://pgp.mit.edu' ...
14 INFO ... 3 keys found
15 INFO key import: 0 imported
16 DEBUG Encrypt the message to the following mixes...
17 DEBUG mixtasy@2.alphatest.mixtasy.net 86FE8E7AB96BB3AB
18 DEBUG mixtasy@3.alphatest.mixtasy.net 4A0460424060FD54
19 DEBUG mixtasy@2.alphatest.mixtasy.net 86FE8E7AB96BB3AB
20 DEBUG mixtasy@1.alphatest.mixtasy.net FE5415106587CDF0
21 DEBUG Random padding of 20223 bytes added to final mix message
22 DEBUG Random padding of 7609 bytes added to intermediate mix message
23 DEBUG Message armored, size increased by 35.8%
24 $ ./sendsmtp.sh mixtasy@1.alphatest.mixtasy.net mixmessage
25 [...]
```

Listing 7.2: Sending a message from server 1.alphatest.mixtasy.net

7.3.2 Mix Message Processing

The `unpack` action (`--unpack`) of the `mixtasy.py` program reads in an intermediate mix message and decrypts its body. Then the verification hash will be calculated and compared with the hash stated in the verification header field. If the revealed message is another intermediate mix message, the body will be padded to the fixed size of 30 kbyte. If the revealed message is a final mix message instead, the original message will be extracted.

As mentioned earlier, Postfix is used to receive and send intermediate mix and original messages. Therefore, the SMTP daemon must be configured to accept mails for the mix address. Postfix' simple content filter functionality [Pos] is used to integrate the Mixtasy service. This is done by setting up a new Postfix service (`mixtasy`) which uses the pipe daemon² to invoke the `mixtasy-filter.sh` script. The address of the sender and receiver (as stated in the SMTP MAIL FROM/RCPT TO command) and the message itself are passed as arguments. The `mixtasy-filter.sh` script passes the message to the Python program if the recipient address matches the mix address of the server. After the message was unpacked, the `mixtasy-filter.sh` script re-injects the unpacked message back to Postfix using the `sendmail` [Venb] command.

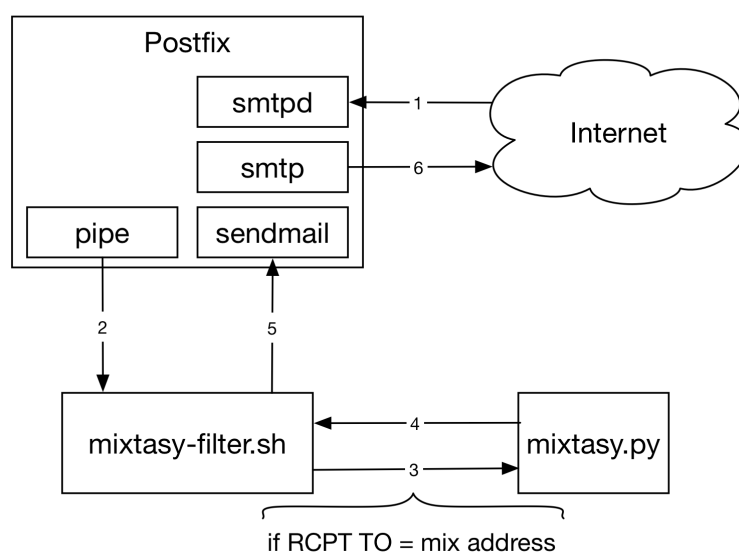


Figure 7.1: Message Processing Pipeline

²“The pipe daemon processes requests from the Postfix queue manager to deliver messages to external commands” [Vena]

Figure 7.1 shows an overview of the message processing pipeline of a mix. Listing 7.3 shows the log messages from the Postfix server while processing the previously send message. The message is received from `localhost` (lines 1-5) and forwarded by the Postfix pipe module to the `mixtasy-filter.sh` script (line 7). Then the `mixtasy-filter.sh` script forwards the message to the `mixtasy.py` program because the address matches the mix servers address. The log output produced by the `mixtasy.py` is showed in Listing 7.4. After the mix message was unpacked it is re-injected to Postfix (Listing 7.3 line 12) and send to the next mix (line 14-19).

```

1 postfix/smtpd: connect from localhost[::1]
2 [...]
3 postfix/qmgr: 1BD4C49876: from=<test@1.alphatest.mixtasy.net>,
4             size=42758, nrcpt=1 (queue active)
5 postfix/smtpd: disconnect from localhost[::1]
6 [...]
7 postfix/pipe: 1BD4C49876: to=<mixtasy@1.alphatest.mixtasy.net>,
8             orig_to=<mixtasy@1.alphatest.mixtasy.net>, relay=mixtasy,
9             delay=0.1, delays=0.01/0/0/0.09, dsn=2.0.0,
10            status=sent (delivered via mixtasy service)
11 postfix/qmgr: 1BD4C49876: removed
12 postfix/qmgr: 3453049884: from=<mixtasy@1.alphatest.mixtasy.net>,
13            size=42572, nrcpt=1 (queue active)
14 postfix/smtp: 3453049884: to=<mixtasy@2.alphatest.mixtasy.net>,
15            orig_to=<mixtasy@2.alphatest.mixtasy.net>,
16            relay=2.alphatest.mixtasy.net [192.168.56:102]:25,
17            delay=1.9, delays=0.35/0.11/0.32/1.1,
18            dsn=2.0.0, status=sent (250 ok)
19 postfix/qmgr: 3453049884: removed

```

Listing 7.3: Postfix log from server 1.alphatest.mixtasy.net

```

1 INFO unpack message...
2 INFO Message is addressed to: mixtasy@1.alphatest.mixtasy.net
3 INFO verification ok
4 INFO Payload is another mix message, addressed to:
5     mixtasy@2.alphatest.mixtasy.net
6 DEBUG Random padding of 674 bytes added to intermediate mix message

```

Listing 7.4: mixtasy.py log from server 1.alphatest.mixtasy.net

7.4 Implementation Specials

This section highlights some parts that are specific for this exact implementation and have not been given by the specification.

7.4.1 Key Retrieval

The key retrieval is not done in an anonymous way (e.g. via the GPG `--use-tor` parameter introduced in GPG version 2.1.10). Instead all keys found at a public key server that match the Mixtasy mix address scheme are imported at startup of the program. A message path is selected on the basis of all local available, valid keys. This means that an adversary admittedly can see if a client uses Mixtasy but cannot get any information about the selected path for a future message.

7.4.2 Prefix Inner Padding

If GPG is used to decrypt a packet which contains a Literal Data Packet followed by random bytes, it extracts the contents in the Literal Data Packet correct but then tries to interpret the following bytes as additional OpenPGP packets. This mostly produces some warnings because the random bytes are of course no valid packets. To avoid the warnings the inner padding, the padding of a final mix message, will be prefixed with two bytes of value `FF 00`. The two bytes represent a OpenPGP new format packet with tag 63 and a zero payload length. Tag 63 is reserved as a private or experimental value in [RFC-4880]. A packet with a reserved tag for private or experimental use is ignored by GPG as the OpenPGP specification advices. GPG also ignores all following data.

7.4.3 Change OpenPGP packet lengths

To change the lengths of the OpenPGP packets contained in a mix message a function was written to parse and rewrite packet headers and lengths. OpenPGP packets can either be in the old or new format. Old format packets support four different length types. The first three are defining different number of bytes used to represent the packet body length. The last packet length type defines an indeterminate packet length (everything until the end of file (eof) belongs to the packet). New format packets also have different variants to define the body length. But instead of an indeterminate packet length partial body lengths are defined. This basically means that one OpenPGP new

format packet could be split into parts glued by a byte sequence representing the length of the next part. For simplicity packets that need to be changed will be converted to the old packet format with an indeterminate packet length.

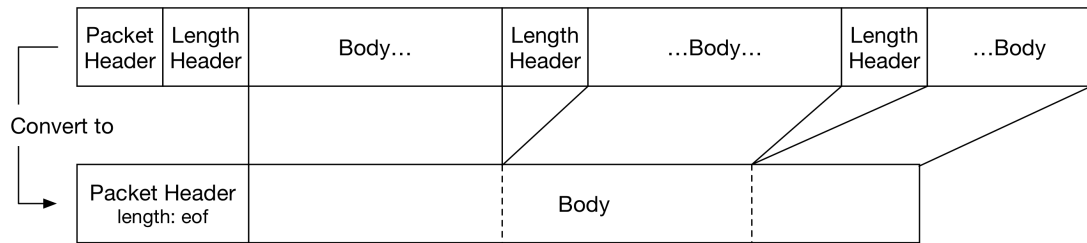


Figure 7.2: OpenPGP Packet Conversion

8 Conclusion

Finally, the objective of this work was satisfied. A specification and a practical prove for a solution, fitting the requirements for secure and anonymous messaging, was shown. The Solution does not require any dedicated services. Every existing Mail Transfer Agent can be upgraded by just installing an extension. This means, the central design goal of easy adoption was reached. The following paragraphs sum up the work in short and give some views to future work and research.

At the beginning of this work the security and privacy problems of plain email communication were stated. Requirements were defined that a solution must fulfill that aims to solve the named problems while keeping the usability properties users are familiar with from using emails. Additionally the solution should have good adoption properties to lower the barrier for wide deployment.

There already exists a great amount of material belonging to secure and privacy preserving messaging in general and email like asynchronous messaging in special. Ranging from add-on solutions to completely new protocols requiring new infrastructure and request the user to readjust their practices. Even it is time to redeem the email protocol family by something completely new with built-in security and privacy features, a temporary solution is required until a successive protocol becomes widely used. Some of the most relevant related projects were described in the Related Work Chapter. While some of them are still under development, others are already technically outdated. But as exposed, none of the solutions met all defined requirements. This led to the decision of specifying a new protocol.

Existing protocols that are operating on top of the email protocols, namely the protocols of the remailer family, were profoundly analyzed in preparation to specify a new protocol. Known weaknesses discovered in the past were considered right from the beginning for the design decisions to prevent them. The defined requirements, especially the claimed adoption properties, were of course also put into account.

The considerations also showed that there is still research required in the domain of

dummy traffic and mixing algorithms. The practical effectiveness of specific dummy traffic policies is not well tested at the moment. Additionally there is little research on practicable ways to make use of client-to-client dummy traffic. Because the choice of the mixing algorithm parameters is always a tradeoff between attack resistance and message latency, a new approach of dynamically changing the parameters on the basis of the message frequency was proposed. But this approach needs to be evaluated in future work before using it in practice.

As the result of the design considerations and the requirement of reusing existing infrastructure and not depending on additional, dedicated services, the Mixtasy protocol was specified. It reuses OpenPGP as a technology for encryption and key management, suggests using existing PGP key servers as a directory service, the Internet Message Format as its data format and SMTP for the transport. This leads to very good conditions to developing well integrable implementations. The usability properties are not completely in the scope of the protocol specification but will depend on the specific implementation of the client software. This should be taken into account when implementing the specification for productive use.

As every protocol that enable sending message anonymously, Mixtasy could potentially be abused (e.g. sending blackmails). But there is at the moment no practicable way known to ban that. And techniques implemented in a protocol that can identify the originator of abuse messages will always weaken the anonymity of the protocol in general. The spam liability of the Mixtasy protocol is expected to be lower than it is for traditional email infrastructure. Firstly because it inherits many of its properties, secondly the cost of encrypting a message is significantly higher than just sending a plain email. Nearly all implicit and explicit spam protection techniques can also be used with Mixtasy. Only content based filters can just be used for unencrypted original messages at exit nodes. Public exit nodes are highly advised to use such techniques to prevent that they get blacklisted.

The protocol does not specify a way for anonymous replies. This leads also to the fact that a mix is not able to deliver a bounce message to the original sender if the message is not deliverable. If a mix cannot correctly process a message, the corresponding bounce message is just send to the last mix because it is the only known address to him. This downside should be mended in the future.

The prototypical implementation was meant to show that the ideas and approaches theoretically described in the specification work in practice. During its development no problems were discovered that would require to redesign the protocol. The operability

was proven by a practical test. Just one fact regarding the scalability was discovered. The most public PGP key servers limit the number of search results to 500¹. To scale well dedicated key servers should be used with no limit on search results.

In addition to the already named topics above some further tasks are open for future work. As already mentioned the implementation is just a prototype and did not cover the whole specification. Therefore, a full, user friendly implementation of the protocol is outstanding. The client could be integrated into existing email clients as a plug-in. An independent detailed evaluation of the specification and future implementations is also be preferable. The OpenPGP perfect forward secrecy extension, which is currently under development, could be integrated to the Mixtasy protocol in future in order to improve the security.

¹500 is the default setting in the most used key server software (SKS, <https://bitbucket.org/sks-keyserver/sks-keyserver/>) for the “Maximum number of matches that will be returned from a query” [Sjo]

List of Figures

3.1	Overall Email Design	11
3.2	OpenPGP Message Format Example	14
4.1	Pseudonymous Remailer Scheme	18
4.2	Mixmaster Remailer Scheme	19
4.3	Mixminion Remailer Scheme	21
6.1	Mixtasy Design: Receiver's provider supports the Mixtasy protocol	34
6.2	Mixtasy Design: Receiver's provider does not support the Mixtasy protocol	34
6.3	Mixtasy Message Format	36
6.4	Intermediate Mix Message Transition	38
6.5	Mixtasy's OpenPGP Key Format	42
7.1	Message Processing Pipeline	51
7.2	OpenPGP Packet Conversion	54

List of Tables

4.1	Requirement Fulfillment of Existing Solutions	23
6.1	Header Field Requirements	39

Bibliography

- [ANSI.X3-4.1986] *Coded Character Set - 7-bit American Standard Code for Information Interchange*. ANSI X3.4. American National Standards Institute, 1986.
- [BL02] Oliver Berthold and Heinrich Langos. “Dummy Traffic Against Long Term Intersection Attacks”. In: *Privacy Enhancing Technologies (PET 2002)*. Ed. by Roger Dingledine and Paul Syverson. Springer-Verlag, LNCS 2482, 2002.
- [Cha81] David Chaum. “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”. In: *Communications of the ACM* 4.2 (1981). URL: <http://www.freehaven.net/anonbib/cache/chaum-mix.pdf>.
- [Cot98] Lance Cottrell. *Mixmaster & Remailer Attacks*. Online. 1998. URL: <http://web.archive.org/web/19981201070355/http://www.obscura.com/~loki/remailer-essay.html> (visited on 04/09/2016).
- [DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. “Mixminion: Design of a Type III Anonymous Remailer Protocol”. In: *2003 IEEE Symposium on Security and Privacy*. May 2003, pp. 2–15. URL: <http://mixminion.net/minion-design.pdf>.
- [DDM07] George Danezis, Roger Dingledine, and Nick Mathewson. *Type III (Mixminion) Mix Protocol Specification*. Draft (“minion-spec.txt”). Apr. 2007. URL: <http://mixminion.net/minion-spec.txt>.
- [DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The Second-Generation Onion Router*. May 2004. URL:

<https://svn.torproject.org/svn/projects/design-paper/tor-design.html>.

- [DP04] Claudia Diaz and Bart Preneel. “Information Security Management, Education and Privacy: IFIP 18th World Computer Congress TC11 19th International Information Security Workshops 22–27 August 2004 Toulouse, France”. In: ed. by Yves Deswarte et al. Boston, MA: Springer US, 2004. Chap. Taxonomy of Mixes and Dummy Traffic, pp. 217–232. ISBN: 978-1-4020-8145-3. DOI: 10.1007/1-4020-8145-6_18. URL: http://dx.doi.org/10.1007/1-4020-8145-6_18.
- [DSD04] Claudia Diaz, Len Sassaman, and Evelyne Dewitte. “Comparison between two practical mix designs”. In: *Proceedings of ESORICS 2004*. LNCS. France, Sept. 2004. ISBN: 978-3-540-22987-2. DOI: 10.1007/b100085. URL: <http://freehaven.net/anonbib/cache/mixmaster-reliable.pdf>.
- [GT96] C. Gulcu and G. Tsudik. “Mixing E-mail with Babel”. In: *Network and Distributed System Security, 1996., Proceedings of the Symposium on*. Feb. 1996, pp. 2–16. DOI: 10.1109/NDSS.1996.492350. URL: <https://gnunet.org/sites/default/files/babel.pdf>.
- [Lev+15] Ladar Levison et al. *Dark Internet Mail Environment Architecture and Specification*. Mar. 2015. URL: <https://darkmail.info/downloads/dark-internet-mail-environment-march-2015.pdf>.
- [Moe+04] U. Moeller et al. *Mixmaster Protocol Version 2*. draft-sassaman-mixmaster-03.txt. Dec. 2004. URL: <https://tools.ietf.org/html/draft-sassaman-mixmaster-03>.
- [Nym99] Jack B. Nymble. *Reliable User’s Manual*. Online. Sept. 1999. URL: <http://www.panta-rhei.dyndns.org/JBnr-en.htm> (visited on 01/16/2015).
- [Pos] *Postfix After-Queue Content Filter*. URL: http://www.postfix.org/FILTER_README.html.

- [RFC-2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. Internet Engineering Task Force, Mar. 1997. URL: <http://www.ietf.org/rfc/rfc2119.txt>.
- [RFC-3207] P. Hoffman. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. Internet Engineering Task Force, Feb. 2002. URL: <http://www.ietf.org/rfc/rfc3207.txt>.
- [RFC-4880] J. Callas et al. *OpenPGP Message Format*. Internet Engineering Task Force, Nov. 2007. URL: <http://tools.ietf.org/html/rfc4880>.
- [RFC-524] J. White. *A Proposed Mail Protocol*. Internet Engineering Task Force, June 1973. URL: <http://www.ietf.org/rfc/rfc524.txt>.
- [RFC-5246] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. Internet Engineering Task Force, Aug. 2008. URL: <http://www.ietf.org/rfc/rfc5246.txt>.
- [RFC-5321] J. Klensin. *Simple Mail Transfer Protocol*. Internet Engineering Task Force, Oct. 2008. URL: <https://tools.ietf.org/html/rfc5321>.
- [RFC-5322] P. Resnick. *Internet Message Format*. Internet Engineering Task Force, Oct. 2008. URL: <http://www.ietf.org/rfc/rfc5322.txt>.
- [RFC-5751] B. Ramsdell and S. Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. Internet Engineering Task Force, Jan. 2010. URL: <http://www.ietf.org/rfc/rfc5751.txt>.
- [RFC-6698] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. Internet Engineering Task Force, Aug. 2012. URL: <http://tools.ietf.org/html/rfc6698>.
- [RFC-draft: PGP PFS] I. Brown, A. Back, and B. Laurie. *Forward Secrecy Extensions for OpenPGP*. Internet Engineering Task Force, Oct. 2001. URL: <http://tools.ietf.org/html/draft-brown-pgp-pfs-03>.

- [RFC-draft: SMTP STS] D. Margolis et al. *SMTP Strict Transport Security draft-margolis-smtp-sts-00*. Internet Engineering Task Force, Mar. 2016. URL: <https://tools.ietf.org/html/draft-margolis-smtp-sts-00>.
- [SDS02] Andrei Serjantov, Roger Dingledine, and Paul Syverson. “From a Trickle to a Flood: Active Attacks on Several Mix Types”. In: *Information Hiding (IH 2002)*. Ed. by Fabien Petitcolas. Springer-Verlag, LNCS (forthcoming), 2002.
- [Sjo] Thomas Sjogren. *SKS - Synchronizing Key Server*. URL: <http://sks.spodhuis.org/sks-man.html>.
- [Syv13] Paul Syverson. “Security Protocols XVII: 17th International Workshop, Cambridge, UK, April 1-3, 2009. Revised Selected Papers”. In: ed. by Bruce Christianson et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. Why I’m Not an Entropist, pp. 213–230. ISBN: 978-3-642-36213-2. URL: http://dx.doi.org/10.1007/978-3-642-36213-2_25.
- [Ung+15] N. Unger et al. “SoK: Secure Messaging”. In: *2015 IEEE Symposium on Security and Privacy*. May 2015, pp. 232–249. URL: <http://cacr.uwaterloo.ca/techreports/2015/cacr2015-02.pdf>.
- [Vena] Wietse Venema. *pipe - Postfix delivery to external command*. URL: <http://www.postfix.org/pipe.8.html>.
- [Venb] Wietse Venema. *sendmail - Postfix to Sendmail compatibility interface*. URL: <http://www.postfix.org/sendmail.1.html>.
- [War12] Jonathan Warren. *Bitmessage: A Peer-to-Peer Message Authentication and Delivery System*. Nov. 2012. URL: <https://bitmessage.org/bitmessage.pdf>.
- [Zhu+05] Ye Zhu et al. “Anonymity analysis of mix networks against flow-correlation attacks”. In: *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005*. Vol. 3. Nov. 2005, 5 pp.–. DOI: 10.1109/GLOCOM.2005.1577959.

Declaration of Originality

German version below

I declare on oath that I completed this work on my own and that information which has been directly or indirectly taken from other sources has been noted as such.

Neither this, nor a similar work, has been published or presented to an examination committee.

Furthermore I declare that the presented copies of this work (printed and digital version) are identical.

Eigenständigkeitserklärung

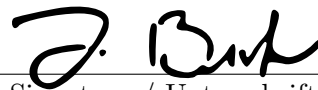
Ich erkläre, dass die vorliegende Masterarbeit von mir selbst verfasst wurde und ich keine anderen als die angeführten Behelfe verwendet, beziehungsweise mich auch sonst keiner unerlaubten Hilfe bedient habe.

Ich versichere, dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Des Weiteren versichere ich, dass die von mir eingereichten Exemplare (ausgedruckt und elektronisch) identisch sind.

Vienna, 21st of May 2016

Place, date / Ort, Datum



Signature / Unterschrift