# MessageVortex

**Transport Independent and Unlinking Messaging**



Inauguraldissertation
zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel
von
Martin Gwerder (06-073-787)
von Glarus GL
May 21, 2020



Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
Auf Antrag von

Prof. Dr. Christian F. Tschudin
Prof. Dr. Heiko Schuldt

Basel, der 18.2.2020 durch die Fakultätsversammlung

_____

Prof. Dr. Martin Spiess

## Abstract

In this paper, we introduce an unobservable message anonymization protocol, named MessageVortex. It bases on the zero-trust principle, a distributed peer-to-peer (P2P) architecture, and avoids central aspects such as fixed infrastructures within a global network. It scores over existing work by blending its traffic into suitable existing transport protocols, thus making it next to impossible to block it without significantly affecting regular users of the transport medium. No additional protocol-specific infrastructure is required in public networks and allows a sender to control all aspects of a message such as the degree of anonymity, timing, and redundancy of the message transport without disclosing any of these details to the routing or transporting nodes. Part of this work is an RFC document attached in Appendix A describing the protocol. It contains all the necessary information to build protocol nodes. The RFC draft is available through the official RFC channels. Additionally, the RFC document, additional documents, and a reference are available under https://messagevortex.net/.

## Acknowledgments

I want to thank my wife Cornelia and my lovely three kids (Saphira, Florian, and Aurelius) for their patience and their support. Without them I could never have done this work.

I want to thank Prof. Dr. C. Tschudin and the University of Basel for the possibility of writing this work and for the challenges they opposed to me, allowing me to grow.

Dr. Andreas Hueni for his thoughts and challenging outside-the-normal-box thinking.

Prof. Dr. Carlos Nicolas of the University of Northwestern Switzerland for being such a valuable sparring partner allowing me to test my ideas.

I want to acknowledge all the individuals who have coded for the LaTeXproject for free. It is due to their efforts that we can generate professionally typeset PDFs (and far more) for free.

# Contents

## List of Corrections

# Part I

# introduction

Almon Brown Strowger was the owner of a funeral parlor in St. Petersburg. He filed a patent on March 10th, 1891 for an "Automatic Telephone Exchange" [**pulseDialingPatent**]. This patent built the base for modern automated telephone systems. According to several sources, he was annoyed by the fact that the local telephone operator was married to another undertaker. She diverted potential customers of Mr. Strowger to her husband instead, which caused Almon B. Strowger to lose business. In 1922, this telephone dialing system, which is nowadays called pulse dialing, became the standard dialing technology for more than 70 years until tone dialing replaced it.

This dialing technology is the base for automatic messaging for voice and text messages (e.g., telex) up until today and is the foundation for current routed networks. These networks build the base for our communication-based Society these days and allow us to connect quickly with any person or company of our wish. We use these networks today as communication meaning for all purposes, and most of the people spend minimal thoughts on the possible consequences arising if someone puts hands on this communication.

This collected data may be used to judge our intentions and thus is not only confidential if we have something to hide. This problem has dramatically increased in the last years as big companies and countries started to collect all kinds of data and created the means to process them. It allows supposedly to judge peoples not only on what they are doing but as well, on what they did and what they might do. Numerous events past and present show that actors, some of which are state-sponsored, collected data on a broad base within the Internet. Whether this is a problem or not is a disputable fact. Undisputed is, however, that such data requires careful handling, and accusations should then base on solid facts. While people may classify personalized advertising as legit use, a general classification of citizens is broadly considered unacceptable[**NCR2013**, **XKeyscore**, **Ball2013**, **Greenberg2013**, **Leuenberger1989**].

To show that this may happen even in democracies, we might refer to events such as the "secret files scandal" (or "Fichenskandal") in Switzerland. In the years from 1900 to 1990 Swiss government collected 900âĂŹ000 files in a secret archive (covering more than 10% of the natural and juristic entities within Switzerland at that time). The Swiss Federal Archives document this event in depth[**Leuenberger1989**].

Whistleblower Edward Snowden leaked a vast amount of documents. These documents suggest that such attacks on privacy are commonly made on a global scale. The documents leaked in 2009 by him claim that there was a data collection starting in 2010. Since these documents are not publicly available, it is hard proving the claims based on these documents. However – A significant number of journalists from multiple countries screened these documents claiming that the information seems credible. According to these documents (verified by NRC), NSA infiltrated more than 50k computers with malware to collect classified or personal information. They furthermore infiltrated Telecom-Operators (mainly executed by British GCHQ) such as Belgacom to collect data and targeted high members of governments even in associated states (such as the mobile phone number of Germany's president) [**NCR2013**, **XKeyscore**, **Ball2013**, **Ackerman2013**, **Greenberg2013**]. A later published shortened list of "selectors" in Germany showed 68 telephone and fax numbers targeting economy, finance, and agricultural parts of the German government. A global survey done by the freedom house[**FOTN2018**] claims a decrease in Internet freedom for the þ8 year in a row.

This list of events shows that big players are collecting and storing vast amounts of data for analysis or possible future use. The list of events also shows that the use of such data was at least partially questionable. This work analyses the possibility of using state-of-the-art technology to minimize the information footprint of a person on the Internet.

We leave a large information footprint in our daily communication. On a regular email, we disclose everything in an "postcard" to any entity on its way. Even when encrypting a message perfectly with today's technology (S/MIME[**RFC2045**] or PGP[**RFC2015**]), it still leaves at least the originating and the receiving entity disclosed, or we rely on the promises of a third party provider which offers a proprietary solution. Even in those cases, we leak pieces of information such as "message subject", "frequency of exchanged messages", "size of messages", or "client being used". A suitable anonymity protocol must cover more than the sent message itself. It includes, besides the message itself, all metadata, and all the traffic flows. Furthermore, a protocol to anonymize messages should not rely on the trust of infrastructure other than the infrastructure under control of the sending or receiving entity. Trust in any third party might be misleading in terms of security or privacy.

Furthermore, central infrastructure is bound to be of particular interest to anyone gathering data. Such control by an adversary would allow manipulating the system or the data or the data flow. So, avoiding a central infrastructure is a good thing when it comes to minimizing an information footprint available to a single entity.

Leaving no information trail when sending information from one person to another is hard to achieve. Most messaging systems disclose at least the peer partners when posting messages. Metadata such as starting and endpoints, frequency, or message size are leaked in all standard protocols even when encrypting messages.

Allowing an entity to collect data may affect senders and recipients of any information. The collection of vast amounts of data allows a potent adversary to build a profile of a person. Unlike in the past, the availability of information has risen to a never known extent with the Internet.

An entity in possession of such Profiles may use them for many purposes. These include service adoption, directed advertising, or classification of citizens. The examples given above show that the effects of this data is not limited to the Internet but reaches us effectively in the real world.

The main problem of this data is that it may be collected over a considerable amount of time and evaluated at any time. It even happened that standard practices at a time are differently judged upon at a later time. Persons may then be judged retrospectively upon these types of practice. This questionable type of judgment is visible in the tax avoidance discussion[**Amat1999**].

People must be able to control their data footprint. Not providing these means does effectively allow any country or a more prominent player to ban and control any number of persons within or outside the Internet.

We design in this work a new protocol. This protocol allows message transfer through existing communication channels. These messages are next to unobservable to any third party. This unobservability does not only cover the message itself but all metadata and flows associated with it. We called this protocol "*MessageVortex*" or just "*Vortex*". The protocol is capable of using a wide variety of transport protocols. It is even possible to switch protocols while the messages are in the transfer. This behavior allows media breaches (at least on a protocol level) and makes the analysis even harder.

The new protocol allows secure communication without the need to trust the underlying transport media. Furthermore, the usage of the protocol itself is possible without altering the immediate behavior of the transport layer. The transport layers' regular traffic does, therefore, increase the noise in which hidden information has to be searched.

# 1 Our Contribution

This thesis contributes to anonymisation with an asynchronous messaging protocol called MessageVortex.

The protocol employs a new type of programmable forwarders called “routing nodes” (nodes) with a novel way of message mixing, moving away from a strictly chunked and onionized system, to a system where routing operations allow to increase or decrease in size without differentiating between decoy traffic and message routing. We refer to the instructions required to process a node as “routing blocks”. These routing blocks have an onionized structure, only exposing the required information for the current node. Routing blocks may travel with a message or join at any common routing node with the message.

To non-traceable routing this, we introduce a novel type of routing operation called “addRedundancy”. This operation is a Reed-Solomon-calculation with encryption and a new type of padding. This operation transposes the received information in a form bigger or smaller than the original message by adding or removing redundancy operations. The applied padding structures the message in such a way that any possible result of a decryption operation results in a plausible padding structure. With standard paddings, decoy operations on traffic would possibly be identifiable as the resulting padding structure may be invalid leaking information. After applying these operations, the routing node then sends this transposed information to subsequent peers without any knowledge of what parts of the sent messages are relevant for the successful message delivery. Therefore, applying such operations makes it impossible for any node to differentiate between decoy traffic and real message traffic. Furthermore, tagging beyond peering nodes is not possible, as building relations between messages of non-neighboring nodes is not possible.

An outside observer is unable to identify messages, as they do not use proprietary communication protocol but hide within other standard internet protocols. We blend these transport protocols without modifying the servers used for message transport. This property makes the protocol very robust as the prosecution of server administrators is not sensible if traffic is running over their infrastructures.

As the structure of routing blocks does not expose the encryption keys required to build routing blocks for a peering node, a malicious node may only discover other possible peer partners when routing traffic without gaining the capability of talking to them. Other properties, such as type of routed traffic, message size, message content, communication partners, or intensity of communication, remain hidden. External global observers are unable to differentiate between regular protocol traffic and Vortex traffic. Assuming an observer capable of identifying the steganographically hidden information, he may apply censorship but remains unable to trace messages according to externally attributes, even assuming that he has additional information from collaborating nodes within the message path.

Our protocol differentiates from other protocols by the fact that our way of mixing and routing messages does not rely on knowingly injected decoy traffic and that we are capable of piggybacking multiple other carrier protocols without modifying the required, already available infrastructure on the internet or requiring dedicated infrastructure. The carrier protocols may even be switched during routing, making it even harder to observe message traffic.

# 2 Scope and Aproach

# 3 Related Research and Systems

## 3.1 Approaches

The main topic of this thesis was defined as follows: 

- Is it possible to have a messaging protocol used on the Internet, based on “state of the science” technologies offering a high level of unlikability (sender and receiver anonymity) towards an adversary with a high budget and privileged access to Internet infrastructure?

Based on this central question, there are several sub-questions grouped around various topics:

1. What technologies and methods may be used to provide sender and receiver anonymity and unlinkability when sending messages against a potential adversary?
2. How can entities utilizing *MessageVortex* be attacked, and what measures are available to circumvent such attacks?
3. How can design mitigate attacks target anonymity of a sending or receiving entity within *MessageVortex*?

**Approach 1: Technologies for Sending Messages Maintaining Unlinkability** This question covers the principal part of the work. We first elaborate on a list of criteria for the *MessageVortex* protocol. We then create a list of suitable technologies and methods. Based on these findings, we define a protocol combining these technologies and researches into a solution. This solution is implemented and analyzed for suitability based on the criteria specified previo 

Main results of this question are found in part **??** and part **??**.

**Approach 2: Attacking unlinkability and circumvention** Within this question, we look at various attacks and test resistance of the protocol based on the definition of the protocol. We do this by first collecting well-known attacks (either generic or specific to a technology used in the protocol). We then elab  if those attacks might be successful (and if so under what circumstances).

We discuss this question in part **??**.

**Approach 3: Attack Mitigation by design** Within this question, we define baselines to mitigate attacks by identifying guidelines for using the protocol. We analyze the effectiveness of the guidelines and elaborate on the general achievement level of the protocol by looking again at the criteria defined in [??].

This question is answered in part **??**.

# 4 Notation

## 4.1 Cryptography

The theory in this document is heavily based on symmetric encryption, asymmetric encryption, and hashing. To use a uniformed notation I use $E^{K_a}(M)$ (where $a$ is an index to distinguish multiple keys) resulting in $\mathbf{M^{K_a}}$ as the encrypted message. If we are reflecting a tuple of information, we write it in boldface. To express the content of the tuple, we use angular brackets $\mathbf{L\langle normalAddress, vortexAddress\rangle}$. If we want Messages encrypted with multiple keys do list the used keys as a comma-separated list in superscript $E^{K_b}\left(E^{K_a}(M)\right) = M^{K_a,K_b}$.

For a symmetric encryption of a message $\mathbf{M}$ with a key $K_a$ resulting in $\mathbf{M^{K_a}}$ where $a$ is an index to distinguish different keys. Decryption uses therefore $D^{K_a}(\mathbf{M^{K_a}}) = \mathbf{M}$.

As notation for asymetric encryption we use $E^{K_a^1}(\mathbf{M})$ where as $K_a^{-1}$ is the private key and $K_a^1$ is the public key of a key pair $K_a^p$. The asymmetric decryption is noted as $D^{K_a^{-1}}(\mathbf{M})$.

For hashing, we do use $H(\mathbf{M})$ if unsalted and $H^{S_a}$ if using a salted hash with salt $S_a$. The generated hash is shown as $H_M$ if unsalted and $H_M^{S_a}$ if salted.

If we want to express what details contained in a tuple we use the the notation $\mathbf{M\langle t, MURB, serial\rangle}$ respectively if encrypted $\mathbf{M^{K_a}\langle t, MURB, serial\rangle}$.

$$
\begin{aligned}
\text{asymetric:} E^{K_a^{-1}}(\mathbf{M}) &= \mathbf{M}^{K_a^{-1}} \\
D^{K_a^1}\left(E^{K_a^{-1}}(\mathbf{M})\right) &= \mathbf{M} \\
D^{K_a^{-1}}\left(E^{K_a^1}(\mathbf{M})\right) &= \mathbf{M} \\
\text{symetric:} E^{K_a}(\mathbf{M}) &= \mathbf{M}^{K_a} \\
D^{K_a}\left(E^{K_a}(\mathbf{M})\right) &= \mathbf{M} \\
\text{hashing (unsalted):} H(\mathbf{M}) &= \mathbf{H}_M \\
\text{hashing (salted):} H^{S_a}(\mathbf{M}) &= \mathbf{H}_M^{S_a}
\end{aligned}
$$

In general, subscripts denote selectors to differentiate the values of the same type, and superscript denotes relevant parameters to operations expressed. The subscripted and superscripted pieces of information are omitted if not needed.

We refer to the components of a *VortexMessage* as follows:

$$
\begin{aligned}
\text{Prefix component:}\mathbf{PREFIX} &= D^{K_a^1}\left(\mathbf{P}^{K_a^{-1}}\right) = D(\mathbf{P}) \\
\text{Header component:}\mathbf{HEAD} &= D^{K_a^1}\left(\mathbf{H}^{K_a^{-1}}\right) = D(\mathbf{H}) \\
\text{Route component:}\mathbf{ROUTE} &= D^{K_a^1}\left(\mathbf{R}^{K_a^{-1}}\right) = D(\mathbf{R})
\end{aligned}
$$

In general, a decrypted Block is written as a capitalized multi-character boldface sequence. An encrypted Block is expressed as a capitalized, single character, boldface letter.

## 4.2 Code and commands

We write code blocks as a light grey block with line numbers:

```
1 public class Hello {
2 public static void main(String args[]) {
3 System.println("Hello␣"+args[1]);
4 }
5 }
```

Commands entered at the command line are in a grey box with a top and bottom line. Whenever root rights are required, the command line is prefixed with a "#". Commands not requiring specific rights are prefixed with a "$". Lines without a trailing "$" or "#" are output lines of the previous command. If long lines are split to fit into the paper, a "↩" is inserted to indicate that a line break was inserted for readability.

```
# su -
# javac Hello.java
# exit
$java Hello
Hello.
$java Hello "This is a very long command-line that had to be broken to fit into the code box displayed on this page."
Hello. This is a very long command-line that had to be broken to fit into the code box displayed on this page.
```

## 4.3 Hyperlinking

The electronic version of this document is hyperlinked. References to the glossary or the literature may be clicked to find the respective entry. Chapter or table references are clickable too.

# 5 Document Structure and Systematics

# Part II

# Concepts and Techniques

# 6 Anonymity Research

In this section, we collect protocols research related to anonymity. We did not stick to anonymous message transfer. Instead, we took a broad focus in terms of technology and outlined in each protocol strengths and weaknesses identified, which may be relevant to this research.

## 6.1 Definition of Anonymity

As the definition for Anonymity we take the definition as specified in [**anonTerminology**].

> Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set.[1]

and

> Anonymity of a subject from an attacker's perspective means that the attacker cannot sufficiently identify the subject within a set of subjects, the anonymity set.[1]

We define the anonymity set as the set of all possible subjects within a supposed message. The anonymity of a subject towards an observing third party is a crucial factor as it relates directly to our adversary model.

## 6.2 $k$-Anonymity

## 6.3 $\ell$-Anonymity

## 6.4 $t$-closeness

## 6.5 Single and Multi Use Reply Blocks

# 7 Censorship

## 7.1 Censorship Resistance

## 7.2 Censorship Circumvention

## 7.3 Parrot Circumvention

# 8 Cryptography and Steganography

[1] footnotes omitted in quote

### 8.1 Homomorphic Encryption

### 8.2 Deniable Encryption

### 8.3 Deniable Steganography

### 8.4 Cryptographic modes for Block Cyphers

### 8.5 Padding for Block Cyphres

## 9 Censorship Circumvention

### 9.1 Technical Forms of Censorship

### 9.2 Zero Trust

## 10 Threat Model

We refer to jurisdiction as a geographical area where a set of legal rules created by a single actor or a group of actors apply, which contains executive capabilities (e.g., police, army, or secret service) to enforce this set of legal rules.

We assume for our protocol that adversaries are state-sponsored actors or players of large organizations. These actors have high funding and expected to have elaborated capabilities themselves or within reach of the sponsor. Actors may join forces with other actors as allies. However, achieving more than 50% on a world scale is excluded from our model. We always assume one or more actors with disjoint interests covering half of the network or more.

We assume the following goals for an adversary:

- An adversary may want to disrupt non-authorized communication.
- An adversary may want to read any information passing through portions of the Internet.
- An adversary may want to build and conserve information about individuals or groups of individuals of any aspect of their life.

To achieve these goals, we assume the following properties of our adversary:

- An adversary has elaborated technical know-how to attack any infrastructure. This attack may cover any attack favoring his goals, starting with exploiting weaknesses of popular software (e.g., buffer overflows or zero-day exploits) down to simple or elaborated (D)DoS attacks.
- An adversary may monitor traffic at any point in public networks within a jurisdiction.
- An adversary may modify routing information within a jurisdiction freely.
- An adversary may freely modify even cryptographically weak secured data where a single or a limited number of entities grant proof of authenticity or privacy.
- An adversary may inject or modify any data on the network of a jurisdiction.

- An adversary may create their nodes in a network. He may furthermore monitor their behavior and data flow without limitation.
- An adversary may force a limited number of other non-allied nodes to expose their data to him. For this assumption, we explicitly excluded actors with disjoint interests.
- An adversary may have similar access to resources as within its jurisdiction in a limited number of other jurisdictions.

we may furthermore subdivide the adversaries into the following sub-classes:

- A censoring adversary
  The primary goal of this adversary is censoring messages and opinions, not within his interests. He does this, regardless of whether the activities of censorship may be observed or not. Therefore, this adversary does not cloak its activities and typically bans censorship circumventing activities as illegal.
- An observing adversary
  This adversary behaves like a traditional spy. He collects and classifies information while hiding its activities. Unlike within reach of a censoring adversary, in this case, typically, no restrictions apply to the use of anonymization technology.

# Part III

# Anonymous Communication Systems

# 11 Well Known Standard Protocols

## 11.1 SMTP and Related Post Office Protocols (1982)

Today's mail transport is mostly done via SMTP protocol, as specified in [**RFC5321**]. This protocol has proven to be stable and reliable. Most of the messages are passed from an MUA to an SMTP relay of a provider. From there, the message is directly sent to the SMTP server of the recipient and subsequently to the server-based storage of the recipient. The recipient may, at any time, connect to his server-based storage and may optionally relocate the message to a client-based (local) storage. The delivery from the server storage to the MUA of the recipient may happen by message polling or by message push (whereas the latter is usually implemented by a push-pull mechanism).

To understand the routing of a mail, it is essential to understand the whole chain starting from a user(-agent) until arriving at the target user (and being read!). To simplify this, we used a consistent model that includes all components (server and clients). The figure 11.1 shows all involved parties of a typical mail routing. It is essential to understand that mail routing remains the same regardless of the client. However, the availability of a mail at its destination changes drastically depending on the type of client used. Furthermore, control of the mail flow and control is different depending on the client.

The model has three main players storage (Storage), agent (Agent) and service (Service). Storages are endpoint facilities storing emails received. Not explicitly shown are temporary storages such as spooler queues or state storages. Agents are simple programs taking care of a specific job. Agents may be exchangeable by other similar agents. A service is a bundle of agents that is responsible for a specific task or task sets.

In the following paragraphs (for definitions), the term "email" is used synonymously to the term "Message". "Email" has been chosen over "messages" because of its frequent use in standard documents.

Emails are typically initiated by a Mail User Agent (MUA). An MUA accesses local email storage, which may be the server storage or a local copy. The local copy may be a cache only copy, the only existing storage (when emails are fetched and deleted from the server after retrieval), or a collected representation of multiple server storages (cache or authoritative).

Besides the MUA, the only other component accessing local email storage is the Mail Delivery Agent (MDA). An MDA is responsible for storing and fetching emails from the local mail storage. Emails destined for other accounts than the current one are forwarded to the MTA. Emails destined to a User are persistently stored in the local email storage. It is essential to understand that email storage does not necessarily reflect a single mailbox. It may as well represent multiple mailboxes (e.g., a rich client-serving multiple IMAP accounts) or a combined view of multiple accounts (e.g., a rich client collecting mail from multiple POP accounts). In the case of a rich client, the local MDA is part of the software provided by the user agent. In the case of an email server, the local MDA is part of the local email store (not necessarily of the mail transport service).

On the server-side, there are usually two components (services) at work. A "Mail Transport Service" (MTS) responsible for mail transfers and a "Mail Storage System" which offers the possibility to store received Mails in a local, persistent store.

An MTS generally consists out of three parts. For incoming connects, there is a daemon called Mail Receiving Agent (Server MRA) is typically a SMTP listening daemon. A Mail Transfer Agent (MTA) which is responsible for routing, forwarding, and rewriting emails. Moreover, a Mail Sending Agent (MSA) which is responsible for transmitting emails reliably to another Server MRA (usually sent via SMTP).

An MSS consists of local storage and delivery agents which do offer uniform interfaces to access the local store. They do also deal with replication issues, and grant should take care of the atomicity of transactions committed to the storage. Typically there are two different kinds of MDAs. Local MDAs offer possibilities to access the store via efficient (non-network based) mechanisms (e.g., IPC or named sockets). This is usually done with a stripped-down protocol (e.g., LMTP). For remote agents there a publicly – network-based – agent available. Common Protocols for this Remote MDA include POP, IMAP, or MS-OXCMAPIHTTP.

Mail endpoints consist typically of the following components:

- A Mail User agent (MUA)
- A Local Mail storage (MUA)
- A Local Mail Delivery Agent (Local MDA)
- A Mail Transfer Agent (MTA)
- A Mail Sending Agent (MSA)
- A Mail Receiving Agent (MRA)

Only two of these components do have external interfaces. These are MSA and MRA. MSA usually uses SMTP as transport protocol. When doing so, there are a couple of specialties.

- Port number is 587 (specified in [**RFC4409**]).
  Although port numbers 25 and 465 are valid and do usually have the same capabilities, they are for mail routing between servers only. Mail endpoints should no longer use them.
- Connections are authenticated.
  Unlike a normal server-to-server (relay or final delivery) SMTP connections on port 25, clients should always be authenticated of some sort. This may be based on data provided by the user (e.g., username/password or certificate) or data identifying the sending system (e.g., IP address)[**RFC4409**]. Failure in doing authentication may result in this port being misused as a sender for UBM.

Mail User Agents (MUA) are the terminal endpoint of email delivery. Mail user agents may be implemented as fat clients on a desktop or mobile system or as an interface over a different generic protocol such as HTTP (Web Clients).

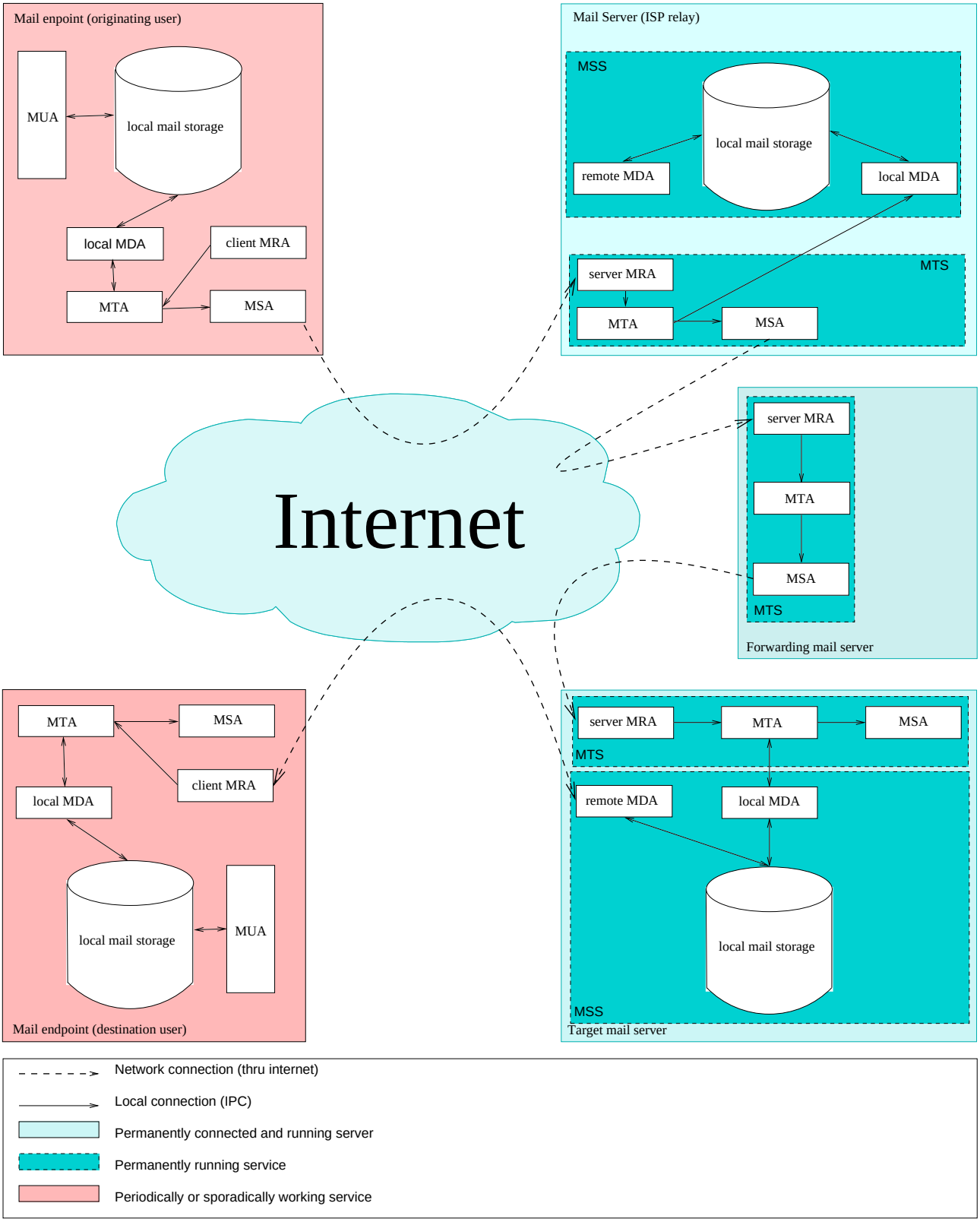


Figure 11.1: Mail Agents

Server located clients are a special breed of fat clients. These clients share the properties of fat clients except for the fact that they do not connect to the server. The client application itself has to be run on the server where the mail storage persists. This makes delivery and communication with the server different. Instead of interfacing with an MSA and a client MDA, they may directly access the local mail storage on the server. On these systems, the local mail storage may be implemented as a database in a user-specific directory structure.

#### 11.1.0.1 Fat clients

The majority of mail clients are fat clients. These clients score over the more centralistic organized web clients in the way that they may offer mail availability even if an Internet connection is not available (through client-specific local mail storage). They furthermore provide

the possibility to collect emails from multiple sources and store them in the local storage. Unlike Mail servers, clients are assumed to be not always online. They may be offline most of the time. To guarantee the availability of a particular email address, a responsible mail server for a specific address collects all emails (the MSS does this) and provides a consolidated view onto the database when a client connects through a local or remote MDA.

As these clients vary heavily, it is mandatory for the MDA that they are well specified. Lack of doing so would result in massive interoperability problems. Most commonly the Protocols IMAP, POP and EWS are being used these days. For email delivery, the SMTP protocol is used.

Fat clients are commonly used on mobile devices. According to [**clientDistribution**] in Aug 2012 the most typical fat email client was Apple Mail client on iOS devices (35.6%), followed by Outlook (20.14%), and Apple Mail (11%). **clientDistribution2**[**clientDistribution2**] as a more recent source lists in February 2014 iOS devices with 37%, followed by Outlook (13%), and Google Android (9%).

#### 11.1.0.2 Server located clients

Server located clients build an absolute minority. This kind of clients was common in the days of centralized hosts. An example for a Server Located Client is the Unix command “mail”. This client reads email storage from a file in the users home directory.

#### 11.1.0.3 Web clients

Web clients are these days a common alternative to fat clients. Most big provider companies use their proprietary web client. According to [**clientDistribution2**] the most common web clients are "‘Gmail"', "‘Outlook.com"', and "‘Yahoo! Mail"'. All these Interfaces do not offer a kind of public plug-in interface. However, they do offer IMAP-interfaces. This important for a future generalistic approach to the problem.

## 11.2 S/MIME (1996)

S/MIME is an extension to the MIME standard. The MIME standard allows in simple text-oriented mails an alternate representation of the same content (e.g., as text and as HTML), or it allows to split a message into multiple parts that may be encoded. It is important to note that MIME encoding is only effective in the body part of a mail.

S/MIME, as described in [**RFC3851**], extends this standard with the possibility to encrypt mail content or to sign it. Practically this is achieved by either putting the encrypted part or the signature into an attachment. It is essential to know that this method leaks significant pieces of the data.

As the mail travels directly from sender to recipient, both involved parties are revealed. Neither message subject nor message size or frequency is hidden. This method does offer limited protection when assuming an adversary with interest in the message content only. It does not protect from the kind of adversary in our case.

The trust model is based on a centralistic approach involving generally trusted root certification authorities.

## 11.3 Pretty Good Privacy (1996)

Exactly as S/MIME PGP[**rfc4880**] builds upon the base of MIME. Although the trust model in PGP is peer-based. The encryption technology does not significantly differ (as seen from the security model).

Like S/MIME, PGP does not offer anonymity. Sender and endpoints are known to all routing nodes. Depending on the version of PGP, some meta-information or parts of the message content such as subject line, the real name of the sender and receiver, message size is leaked.

A good thing to learn from PGP is that peer-based approaches are offering limited possibilities for trust. The trust in PGP is based on the peer review of users. This peer review may give an idea of how well verified the key of a user is.

# 12 Information Routing and Distribution for Anonymizing Protocols

## 12.1 Mixing

Mixes have been first introduced by **CHAUM1**[**CHAUM1**] in **CHAUM1**. The basic concept in a mix goes as follows. We do not send a message directly from the source to the target. Instead, we use a kind of proxy server or router in between which picks up the packet, anonymizes it, and forwards it either to the recipient or another mix. If we assume that we have at least three mixes cascaded, we then can conclude that:

- Only the first mix knows the true sender
- All intermediate mixes know neither the true sender nor the true recipient (as the data comes from mixes and is forwarded to other mixes)
- Only the last mix knows the final recipient.

This approach (in this simple form) has several downsides and weaknesses.

- In a low latency network, the message may be traced by analyzing the timing of a message.
- We can emphasize a path by replaying the same message multiple times (assuming we control an evil node), thus discovering at least the final recipient.
- If we can “tag” a message (with content or attribute), we then may be able to follow the message.

In **RP03-1 RP03-1** analyzed the suitability for mixes as an anonymizing network for masses. They concluded that there are three possibilities to run mixes.

- Commercial, static MixNetworks
- Static MixNetworks operated by volunteers
- Dynamic MixNetworks

They concluded that in an ideal implementation, a dynamic mix network where every user is operating a mix is the most promising solution as static mixes always might be hunted by an adversary.

## 12.2 Anonymous Remailers

Remailers have been in use for quite some time. There are several classes of remailers, and all of them are somehow related to Mixnets. There are “types” of remailers defined. Although these “types” offer some hierarchy, none of the more advanced “types” seem to have more than one implementation in the wild.

Pseudonymous Remailers (also called Nym Servers) take a message and replace all information pointing to the original sender with a pseudonym. This pseudonym may be used as an answer address. The most well known pseudonymous remailer possibly was anon.penet.fi run by Johan Helsingius. This service has been forced several times to reveal a pseudonyms true identity before Johan HeÃűsingius decided to shut it down. For a more in-depth discussion of Pseudonymous Remailers see 13.2.1

Cypherpunk remailers forward messages like pseudonymous remailers. Unlike pseudonymous remailers, Cypherpunk remailers decrypt a received message, and its content is forwarded without adding a pseudonym. A reply to such a message is not possible. They may, therefore, be regarded as an “decrypting reflector” or a “decrypting mix” and may be used to build an onion routing network for messages. For a more in-depth discussion of type-1-remailers, see section 13.2.2.

Mixmaster remailers are very similar to Cypherpunk remailers. Unlike them, Mixmaster remailers hide the messages, not in an own protocol, but use SMTP instead. While using SMTP as a transport layer, Cypherpunk remailers are custom (non-traditional mail) servers listening on port 25. For a more in-depth discussion of type-2-remailers, see section 13.2.4.

Mixminion remailers extend the model of Mixmaster remailers. They still use SMTP but introduce new concepts. New concepts in Mixminion remailers are:

- Single Use Reply Blocks (SURBs)
- Replay prevention
- Key rotation
- Exit poicies
- Dummy traffic

For a more in depth discussion of Mixminion remailers see section 13.2.8.

## 12.3 Onionizing

Onion routing is a further development of the concept of mixes. In onion routers, every mix gets a message which is asymmetrically encrypted. By decrypting the message, he gets the name of the next-hop and the content which he has to forward. The main difference in this approach is that in traditional mix cascades, the mix decides about the next hop. In an onionised routing system, the message decides about the route it is taking.

While tagging attacks are far harder (if we exclude side-channel attacks to break sender anonymity), the traditional attacks on mixes are still possible. So when an adversary is operating entry and exit nodes, it is straightforward for them to match the respective traffic.

One very well known onion routing network is Tor (https://www.torproject.org). For more information about Tor see section 13.2.6.

## 12.4 Crowds

Crowds is a network that offers anonymity within a local group. It works as follows:

- All users add themselves to a group by registering on a so-called “blender”.
- All users start a service (called JonDo).

| | Topology |
|---|---|
| MessageVortex | ⊠ |
| Riffle | |
| Atom | |
| Riposte | |
| Pung | |
| PIR | |
| Karaoke | |
| Loopix | |
| Stadium | |
| Vuvuzela | |

Table 13.1: Classification table for anonymization protocols according to [**Shirazi2018**]

- Every JonDo takes any received message (might be from him as well) and sends it with a 50% chance either to the correct recipient or to a randomly chosen destination

While crowds as specified in [**crowds:tissec**] does anonymize the sender from the recipient rather well, the system offers no protection from someone capable of monitoring crowds traffic. The system may, however, be easily attacked from within by introducing collaborating johndos. It has been further developed to D-Crowds [**DBLP:conf/esorics/DanezisDKT09**], ADU/RADU [**Munoz-Gea2008**], Freenet[**freenet**] and others.

Furthermore, the blender is aware of all JonDos and thus of particular interest for any observing or censoring adversary. Control of the blender enables an adversary to split the network into controllable parts, adding a high likelihood of discovering an original sender.

## 12.5 Mimic Routes

Mimics are a set of statical mixes which maintain a constant message flow between the static routes. If legitimate traffic arrives, the pseudo traffic is replaced by legitimate traffic. An outstanding observer is thus incapable of telling the difference between real traffic and dummy traffic.

If centralized mixes are used, the system lacks the same vulnerabilities of sizing and observing the exit nodes as all previously mentioned systems. If we assume that the sender and receiver operate a mixer by themselves, the system would no longer be susceptible to timing or sizing analyses. The mimic routes put a constant load onto the network. This bandwidth is lost and may not be reclaimed. It does not scale well as every new participant increases the need for mimic routes and creates (in the case of user mixes) a new mimic load. Furthermore, the mixes are easily identifiable as their characteristic data stream contrasts compared to other network service streams.

## 12.6 Distributed Hash Tables

## 12.7 Dining Cryptographer Networks

DC networks are based on the work **chaum-dc** by **chaum-dc**[**chaum-dc**]. In this work, **chaum-dc** describes a system allowing a one-bit transfer (The specific paper talks about the payment of a meal). Although all participants of the DC net are known, the system makes it unable to determine who has been sending a message. The message in a DC-Net is readable for anyone. This network has the downside that a cheating player may disrupt communication without being traceable.

Several attempts have been made to strengthen the proposal of Chaum[**golle:eurocrypt2004**, **disco**, **herbivore:tr**, **Corrigan-Gibbs:2010:DAA:1866307.1866346**]. However, no one succeeded without introducing significant downsides on the privacy side.

# 13 Proposed Academic Protocols and System Implementations

## 13.1 Characteristics of Known Anonymity Implementations

Table **??** shows the previously analyzed protocols.

## 13.2 Resenders, Onion Routers, and MixNets

### 13.2.1 Pseudonymous Remailers (1981)

### 13.2.2 Cypherpunk Remailers (approx. 1993)

With the failing of anon.penet.fi, it became clear that the weakest spot of a single server infrastructure the information stored on the server and the vulnerability of their owner. The new type-1-remailers score over the existing type-0-remailers by using encryption for the message. Most of the time PGP was used and custom programmed mail processors on systems to achieve the functionallity. It is unclear when first type-1-remailers were invented. Setting up a type-1-remailer was typically achieved by using procmail together with a small script calling PGP binaries and then sending the resulting message to the next recipient. By combining multiple type-1-remailers, an onion-like structure of the message was achievable.

This approach was promising, but it was still observable. An observation was possible by correlating the message sizes (e.g., strictly decreasing) and timing information. Furthermore, remailers were however still known and authorities were able to ban infrastructure and capable of monitoring their routing activities. Additionally, those remailers allowed to prosecute administrators of such systems.

### 13.2.3 Babel (1996)

Babel was an academic system defined in a paper by **babel** in **babel**[**babel**]. It has been developed at IBM Zurich Research Laboratory. It was a mixing system using onionized addresses. The sender remains anonymous while he may provide a reply routing block called RPI. If both parties would like to remain anonymous, the RPI of the initiator is deployed in a forum thread. Anyone using this block adds an RPI for its address to the message.

This system has all the disadvantages of a system using MURBs. Traffic highlighting and similar attacks are possible.

### 13.2.4 Mixmaster-Remailers (1996)

Like Cypherpunk remailers, the Mixmaster remailers were working with onion-like encrypted messages. The protocol was based on Mix-Nets described by Chaum in [**CHAUM1**] and further developed by L. Cotrell in 1996.

In contrast to type-1-remailers, the use of cascading systems to remail became systematic. The enduser used specialized software to build and send Mixmaster messages.

Mixmaster messages were still traceable by message size. Reply blocks were not supported by the system. A user had to know all Mixmaster nodes in order to use the system. The last node was typically an exit node sending the message in clear to the final recipient. This behavior still allowed the use of Usenet.

### 13.2.5 Crowds (1997)

### 13.2.6 Tor (2000)

Tor is one of the most common onion router networks these days and onionizes generic TCP streams. It is specified in [**tor-spec**]. It might be considered one of the most advanced networks since it has a considerable size, and much research has been done here.

According to [**onion-routing:pet2000**] Tor is a network consisting of multiple onion routers. Each client first picks an entry node. Then it establishes an identity, gets a listing of relay servers, and chooses a path through multiple onion routers. The temporary identity links to such a path and should be changed on a regular base along with its identity. Transferring data works by splitting the data into equally sized cells of 512 bytes.

There is a centrally organized directory in the Tor network, knowing all tor relay servers. Any Tor relay server may be a directory server as well.

Many attacks involving the Tor networks have been discussed in the academic world such as [**hs-attack06**, **esorics13-cellflood**, **bauer:wpes2007**, **esorics12-torscan**, **oakland2013-trawling**, **danner-et-al:tissec12**, **congestion-longpaths**] and some have even been exploited actively. In the best case, the people discovering the attacks did propose mitigation to the attack. Some of these mitigations flowed back into the protocol. Some general thoughts of the attacks should be emphasized here for treatment in our protocol.

Being an exit node may be a problem in some jurisdictions. In general, it seems to be accepted that routing traffic with unknown content (to the routing node) is not regarded as illegal per se. So by being unable to tell malicious or illegal traffic apart from legitimate traffic, this is not a problem. However – being an exit node can mean that unencrypted and illegal traffic is leaving the routing traffic. In this specific case, operators of a relay node might fear legal prosecution. Tor nodes may proclaim themselves as " non-exit nodes" to avoid the possibility of legal prosecution.

Furthermore, several DoS-Attacks have been carried out to overload parts of the Tor network. Most of them do a bandwidth drain on the network layer.

Attacking anonymization has been done in several ways. First of all, the most common attack is a time-wise correlation of packets if in control of an entry and an exit node. A massive attack of this kind was published in 2014 and has been published on the tor website (relay early traffic confirmation attack). This attack was possible because tor is a low latency network. Another attack is to identify routes through tor by statistically analyze the traffic density in the network between nodes. More theoretical attacks focus on the possibility of controlling the directory servers to guarantee that an entity may be deanonymized because it is using compromised routers.

Generally, the effectiveness of the monitoring of single nodes or whole networks is disputed. According to a study by **ccs2013-usersrouted** in **ccs2013-usersrouted**[**ccs2013-usersrouted**], a system in the scale of PRISM should be able to correlate traffic of 95% of the users within a "few days". Other sources based on the Snowden Papers claim that NSA was unable so far to de-anonymize users of Tor. However, since these papers referenced to "manual analysis", the statement may be disputed when looking at automated attacks as well.

It is, according to `https://www.torproject.org/docs/pluggable-transports`, impossible to use transborder Tor traffic in at least China, Uzbekistan, Iran, and Kazakstan. In censored countries, Tor offers so-called bridged Transports. Currently deployed transports in the standard Tor browser bundle package are obfs4, meek, FTE, and ScrambleSuit. Only meek is listed as working in China. Meek achieves this by hiding its traffic in a standard protocol (https).

[**saleh2018shedding**] is an excellent survey listing recent developments and attacks within the Tor project.

### 13.2.7 $I^2P$ (2001)

The name $I^2P$ is a derived from "Invisible Internet Project" according to geti2p.net. The first binary relase on sourceforge dates from 2001. The system itself is comparable to Tor for its capabilities. Mayor differences are:

- P2P based
- Packet-switched routing (tor is "circuit-switched")
- Different forward and backward routes (called tunnels)
- Works pseudonymously
- Supports TCP and UDP

$I^2P$ has not attracted as much attention as Tor so far. So it is hard to judge upon its real qualities.

In **pets2011-i2p pets2011-i2p** presented in [**pets2011-i2p**] an attack. As $I^2P$s security model is chosen based on IP addresses, the authors propose to use several cloud providers in different B-Class networks. By selectively flooding peers, an adversary may extract statistical information. The paper proposes an attack based on the heuristic performance-based peer selection. The main critics of the paper were that the peer selection might be influenced by an adversary enabling him to recover $I^2P$ has not attracted as much attention as Tor so far. So it is hard to judge upon its real qualities.

In **pets2011-i2p pets2011-i2p** presented in [**pets2011-i2p**] an attack. As $I^2P$s security model is chosen based on IP addresses, the authors propose to use several cloud providers in different B-Class networks. By selectively flooding peers, an adversary may extract statistical information. The paper proposes an attack based on the heuristic performance-based peer selection. The main critics of the paper were that the peer selection might be influenced by an adversary enabling him to recover data on a statistical base.

### 13.2.8 Mixminion-Remailers (2002)

Mixminion was the standard implementation of a type-3-remailer. It tried to address many issues previously not solved. A Mixminion router splits messages in equally sized chunks and supports SURBs. Furthermore, replay protection and key rotation were available. Unlike the previous remailer types, Mixminion was no longer using SMTP as the transport protocol. Instead, Mixminion introduced a new transport protocol. The sources of this remailer are available on GitHub under https://github.com/mixminion/mixminion.

As a received message had to be decoded by the final recipient. Therefore, the final recipient had to be aware of Mixminion system.

According to `https://mixminion.net` the first release of the software was in December 2002. And has been discontinued in 2008. Since 2011 the sources are available on GitHub. Therehave been some forks in 2011 but at the moment all forks seem to be inactive since at least 2016 as there are no new commits.

### 13.2.9 $\mathcal{P}^5$ (2002)

The Peer-to-Peer Personal Privacy Protocol is defined in [**sherwood-protocol**]. It provides sender-, receiver- and sender-receiver anonymity. According to the project page of $\mathcal{P}^5$, there is only a simulator available for the protocol.

The transport layer problematic has been wholly ignored. As there is no precise protocol specification but only a rough outline about the messaging and the crypto operations, $\mathcal{P}^5$ offers minimal possibilities for analysis.

### 13.2.10 AN.ON (2003)

AN.ON, as suggested in [**federrath2003system**], is a mixing network. It generates messages in equally sized chunks and sends them in fixed time slots after random mixing. Its implementation is called JAP and may be found under https://anon.inf.tu-dresden.de/. JAP is many ways similar to the capabilities of Tor. The network was at the time of writing a lot smaller (10 JonDos compared to 6500 relays in the Tor network).

While the approach is both simple and effective, it is not suitable against a powerful adversary. First, an adversary may be able to snoop the forwarding when on the system. Second, due to the timing behavior, tunnels belonging to each other may be identified, and third, the package size information does leak as well.

### 13.2.11 AP3 (2004)

AP3, as defined in [**mislove2004ap3**], is an anonymous communication system and very similar to crowds. It performs a random walk over a set of known nodes. Not all nodes are known to anyone, and all nodes are aware of the final recipient.

The system is susceptible to numerous attacks, as shown by [**ccs2008:mittal**], and does not withstand our adversary as the final recipient is known to the routing nodes.

### 13.2.12 Cashmere (2005)

Cashmere is specified in [**zhuang2005cashmere**]. It defines a protocol for the use of chaum mixes. Unlike most of the protocols, the chaum mixes in cashmere are virtual. So-called relay groups represent them. Each mix in the relay group may be used as an equivalent mix to all other mixes in the same group.

This design means that the failure of one mix does not result in the non-delivery of a message.

No client implementation could be found on the nternet. The project homepage http://current.cs.ucsb.edu/projects/cashmere/ has not been updated since 2005. This suggests that this project is dead or sleeping.

### 13.2.13 SOR (2012)

SSH-based onion routing (SOR)[**Egners_2012**] is blaming the complex and monocultural landscape of anonymizing software and proclaims a simple approach based on onionized SSH tunnels.

### 13.2.14 SCION (2017)

SCION[**perrig2017scion**] is a clean slate Internet protocol. While SCION is not really an anonymizing protocol. It contains, however, many interesting features. Unlike with the traditional networks, we have the possibility of influencing the routing of data within SCION. Furthermore, with PHI[**chen2017phi**] and Dovetail[**sankey2014dovetail**], SCION may feature strong and fast anonymity features.

Unfortunately, as this is a clean slate Internet design, it is not available commonly currently, and as it is easily identifiable, it enables easy censorship as the relevance is due to its current availability of no importance, and a censoring adversary may just ban and censor SCION entirely.

## 13.3 Distributed Hash Tables

### 13.3.1 Tarzan (2002)

Tarzan is a P2P IP protocol using UDP to communicate. It is specified in [**tarzan:ccs02**]. Tarzan nodes may be used to anonymize Internet traffic in general. An initiator on the original sender machines encapsulates traffic into a layered UDP package and sends the package through a mix like relayd's. The last relayd acts as an exit node. A replier may send answers the opposite way. Each relayd knows its next and previous relayd. To minimize the impact of observation, Tarzan forwards packets only every 20ms and features replay protection.

### 13.3.2 MorphMix (2002)

MorphMix is another mix network and specified in [**morphmix:wpes2002**]. It was a circuit-based mix system for networking anonymity. The core of the network was collision detection. This detection has been circumvented by [**morphmix:pet2006**]. Since then, no new papers have been published, and the project seems to be dead.

### 13.3.3 Salsa (2008)

Salsa was proposed in [**Salsa**] and described a circuit based anonymization pattern based on distributed hash tables (DHT). An implementation for Salsa is available, but it is not public. [**ccs2008:mittal**] claims that by combining active and passive attacks, anonymity can be compromised.

## 13.4 Dining Cryptographer Based Network

### 13.4.1 Herbivore (2003)

Herbivore is a network protocol designed by **herbivore:tr** in [**herbivore:tr**]. It is based on the dining cryptographers paper[**chaum-dc**]. At the time of writing, no herbivore client or an actual protocol implementation could be found on the Internet. Wikipedia lists Herbivore as "dormant or defunct".

### 13.4.2 Dissent (2010)

Dissent is defined in [**Corrigan-Gibbs:2010:DAA:1866307.1866346**]. It is an anonymity network based on DC-nets. A set of servers forms these DC-nets. At least one of the servers in the used net must be trustworthy, and none may be misbehaving. A server failure results in the stall of all message delivery using this server.

## 13.5 Broadcast and Multicast Networks

### 13.5.1 Hordes (2002)

Hordes was a multicast-based protocol for anonymity specified in [**Levine:2002**]. Hordes used the abilities to handle multicast addresses of routers to generate a dynamic set of receivers and then sends messages to them. It assumes that a single observer or router does not know all participating peers.

This assumption is correct for a local observer. Unfortunately, it is not sufficient assuming an adversary as defined in this paper.

## 13.6 Distributed Storage Systems

### 13.6.1 Feenet (2000)

Freenet was initially designed to be a fully distributed data store[**freenet**]. Documents are stored in an encrypted form. Downloaders must know a document descriptor called CHK containing the file hash, the key, and some background about the crypto being used. A file is stored more or less redundantly based on the number of accesses to a stored file. The primary goal of Freenet is to decouple authorship from a particular document. It furthermore provides fault-tolerant storage, which improves caching of a document if requested more often.

Precisely as $I^2P$, Freenet is not analyzed thoroughly by the scientific world.

The Freenet features two protocols FCPv2 acts as the client protocol for participating in the control of the Freenet storage. The Freenet client protocol allows us to insert and retrieve data, to query the network status, and to manage Freenet nodes directly connected to an own node. FCPv2 operates on port 9481, and blocking is thus easy, as it is a dedicated port.

The Freenet project seems to be under active development as pages about protocols were updated in the near past (Last update on the FCPv2 Page was July 5$^{th}$ 2016 at the time of writing).

### 13.6.2 Gnutella (2000)

Gnutella is not a protocol for the anonymity world in special. Instead, the Gnutella protocol implements a general file sharing on a Peer to peer base. This peer-to-peer approach is the most interesting aspect of Gnutella in this context. Furthermore, Gnutella has proven to be working with a large number of clients.

The current protocol specification may be found under http://rfc-gnutella.sourceforge.net/. While the Gnutella network is defunct. The approaches solving some of the peer-to-peer aspects were very interesting.

### 13.6.3 Gnutella2 (2002)

Despite its name, Gnutella2 is not the next generation of Gnutella. It was a fork in 2002 from the original Gnutella and has been developed in a different direction. The specification may be found on `http://g2.doxu.org`. Just as its predecessor, Gnutella2 seems to be dead. The last relevant update to the main site or its protocol is dated four years back.

## 13.7 Unknown (TBD)

### 13.7.1 Riffle (2016)


[**2016riffle**]

### 13.7.2 Atom (2017)


[**2017atom**]

### 13.7.3 Riposte (2015)


[**gan2015riposte**]

### 13.7.4 Pung (2016)


[**2016unobservable**]

| | Topology |
|---|---|
| MessageVortex | ⊠ |
| Riffle | |
| Atom | |
| Riposte | |
| Pung | |
| PIR | |
| Karaoke | |
| Loopix | |
| Stadium | |
| Vuvuzela | |

Table 13.2: Classification table for anonymization protocols according to [**Shirazi2018**]

### 13.7.5 PIR (2018)

**l2018pir**]

### 13.7.6 Karaoke (2018)

**2018karaoke**]

### 13.7.7 Loopix (2017)

**owska2017loopix**]

### 13.7.8 Stadium (2017)

**2017stadium**]

### 13.7.9 Vuvuzela (2015)

**015vuvuzela**]

### 13.7.10 Characteristics of known anonymity Systems

Table **??** shows the previously analyzed protocols.

## 13.8 Pseudo Random Number Generators

The following sections list two PRNG specifications to follow the recommendations of [**rfc1750**]. These PRNGs are used to complete the padding specified in the addRedundancy operation.

We have chosen to support two kinds of PRNG. These algorithms are not relevant for the security of the system, but they guarantee non-detectable padding when doing the addRedundancy operation. The two PRNGs selected were xorshift128+ and Blum Micail PRNG. Both PRNGs were quoted to pass BigCrush. However, recent development shows that this might not be true for xorshift128+, as demonstrated in [**LEMIRE2019139**].

## 13.9 Known Attacks

In the following sections, we emphasize on possible attacks to an anonymity preserving protocols. These attacks may be used to attack the anonymity of any entity involved in the message channel. In a later stage, we test the protocol for immunity against these classes of attacks.

### 13.9.1 Broken Encryption Algorithms

Encryption algorithms may become broken at any time. This either to new findings in attacking them, by more resources being available to an adversary, or by new technologies allowing new kinds of attacks. A proper protocol must be able to react to such threads promptly. This reaction should not rely on a required update of the infrastructure. Users should solely control the grade of security.

We cannot do a lot for attacks of this kind to happen. However, we might introduce a choice of algorithms, paddings, modes, and key sizes to give the user a choice in the degree of security he wants to have.

### 13.9.2 Attacks Targeting Anonymity

Attacks targeting users anonymity are the main focus of this work. Many pieces of information may be leaked, and the primary goal should, therefore, rely on the principles established in security.

- Prevent an attack
  Attack prevention can only be done for attacks that are already known and may not be realistic in all cases. In our protocol, we have strict boundaries defined. A node under attack should at any time of protocol usage (this excepts bandwidth depletion attacks) be able to block malicious identities. Since establishing new identities is costly for an attacker, he should always require far more resources than the defender.
- Minimize attack surface
  This part of the attack prevention is included by design in the protocol.
- Redirect an attack
  Although the implementation does not do this, it is possible to handle suspected malicious nodes differently.
- Control damage
  For us, this means leaving as little information about identities or meta information as possible on untrusted infrastructures. If we leave traces (i.e., message flows, or accounting information) they should have the least possible information content and should expire within a reasonable amount of time.
- Discover an attack
  The protocol is designed in such a way that attack discovery (such as a query attack) is possible. However, we consider active attacks just as part of the regular message flow. The protocol must mitigate such attacks by design.
- Recover from an attack
  An attack does always impose a load onto a system's resources regardless of its success. It is vital that a system recovers almost immediately from an attack and is not covered in a non-functional or only partial-functional state either temporarily or permanently.

In the following subsections, we list a couple of attack classes that have been used against systems listed in **??** or the respective academic works. We list the countermeasures which have been taken to deflect these attacks.

#### 13.9.2.1 Probing Attacks

Identifying a node by probing and check their reaction is commonly done when fingerprinting a service. As a node is participating in a network and relaying messages probing may not be evaded. However, it may be made costly for an adversary to do systematic probing. This should be taken into account. Both currently specified transport protocol features an indefinite number of possible accounts. Since not the server but the endpoint address is behaving, node probing is more complicated than in other cases where probing of service is sufficient.

One of the problems is clear-text requests. These requests may be used on any transport layer account without previous knowledge of any host key. Thus the recommendation in table **??** is generally not to answer the requests. Routing nodes in jurisdictions not fearing legal repression or prosecution may reply to clear text requests, but it is usually discouraged as they allow harvesting of addresses.

One strategy to avoid would be to put high costs onto clear-text requests in such a way that a clear-text request may have a long reply time (e.g., up to one day). A node is free to blacklist an identity in case of an early reply. This is an insufficient strategy as a big adversary may have lots of identities in stock. Requesting an unusually long key as a plain-text identity does not make sense either as these as well may be kept in stock. We may, however, force a plaintext request to have an identity block with a hash following specific rules. We may, for example, put in a requirement that the first four bytes of the hash of a header block translates to the first four characters of the routing block. At the moment, this has been rejected in the standard for practical reasons. First, as the request is unsolicited, a sender is the only one able to decide the algorithm of the hash. This would allow a requester to choose upon the complexity of the puzzle. Second, any negotiation of the cost of the request would result in the disclosure of the node as VortexNode, which might be unsuitable.

#### 13.9.2.2 Hotspot Attacks

Hotspot attacks aim to isolate high traffic sites within a network. By analyzing specific properties or the general throughput locations with outstanding traffic may be identified. These messages do quite often reveal senders or recipients. Sometimes an intermediate node in an anonymizing system.

#### 13.9.2.3 Message Tagging and Tracing

When using an anonymization system, a message may be either fully or partially traced or even tagged. Tagging allows one to recognize a message at a later stage and map it to its predecessors. Protocols with tagable messages are not suitable for anonymization systems.

#### 13.9.2.4 Side Channel Attacks

Side-channel attacks are numerous. Especially important to us are attacks related to either lookup in independent channels (e.g., downloading of auxiliary content of a message) or behavior related to timing patterns.

#### 13.9.2.5 Sizing Attacks

There are two kinds of sizing attacks identified to be relevant for us. One is the possibility for matching messages with related sizes, and the other one is to relate message size to the original messages. Both attacks may be considered as a tracing attack and will be analyzed accordingly.

#### 13.9.2.6 Bugging Attacks

Numerous attacks are available through the bugging of a protocol. In this chapter, we outline some of the possibilities and how they may be countered:

- Bugging through certificate or identity lookup:
  Almost all kinds of proof of identities, such as certificates, offer some revocation facility. While this is a perfect desirable property of these infrastructures, they offer a flaw. Since the location of this revocation information is typically embedded in the proof of identity, an evil attacker might use a falsified proof of identity with a recording revocation point.

  There are multiple possibilities to counter such an attack. The easiest one is to do no verification at all. Having no verification is, however, not desirable from the security point of view. Another possibility is only to verify trusted proof of identities. By doing so, the only attacker could be someone having access to a trusted source of proof of identities. A third possibility is to relay the request to another host either by using an anonymity structure such as Tor or by using its infrastructure. Using Tor would violate the “Zero Trust” goal. Such a measure would only conceal the source of the verification. It would not hide the fact that the message is processed. A fourth and most promising technology would be to force the sender of the certificate to include a “proof of non-revocation”. Such a proof could be a timestamped and signed partial CRL. It would allow a node to verify the validity of a certificate without being forced to disclose itself by doing a verification. On the downside has to be mentioned that including proof of non-revocation involves the requirement to accept a certain amount of caching time to be accepted. This allowed caching time reduces the value of the proof as it may be expired in the meantime. It is recommended to keep the maximum cache time as low as 1d to avoid that revoked certificates may be used.

- Bugging through DNS traffic:
  A standard protocol on the Internet is DNS. Almost all network-related programs use it without thinking. Typically the use of such protocol is only a minor issue since the resolution of a lookup usually done by an ISP. In the case of a small Internet service provider (ISP), this might, however, already become a problem.

  The bugging in general attack works as follows: We include a unique DNS name to be resolved by a recipient. This can be done most easily by adding an external resource such as an image. A recipient will process this resource and might, therefore, deliver information about the frequency of reading, or the type of client.

  It must be taken into account that the transport layer will always do DNS lookups and that we may not avoid this attack completely. We may, however, minimize the possibilities of this attack.

- Bugging through external resources:
  A straightforward attack is always to include external resources into a message and wait until they are fetched. In order to avoid this kind of attack, plain text or other self-contained formats should be used when sending a message. As we may not govern the type of contained message, we can make at least recommendations concerning its structure.

### 13.9.3 Denial of Service Attacks

#### 13.9.3.1 Censorship

Whereas traditional censorship is widely regarded as selective information filtering and alteration, very repressive censorship can even include denial of information flows in general. Any anonymity system not offering the possibility to hide in legitimate information flows, therefore not censorship-resistant.

#### 13.9.3.2 Denial of service

An adversary may flood the system in two ways.

- He may flood the transport layer exhausting resources of the transport system.
  This is a straightforward attack. MessageVortex has no control over the existing transport protocol. Therefore, all flooding attacks on that layer are still effective. However, If an adversary attacks a node, the redundancy of a message may still be sufficient. On the other hand, flooding disrupts at least all other services using the same transport layer on that node. This result may be unacceptable for an attacker. More likely would be censorship.

- He may flood the routing layer with invalid messages.
  Identifying the messages is relatively easy for a node. Usually, it should be sufficient to decode the CPREFIX block of a message. If the CPREFIX is valid, then the header block either identifies a valid identity or processing may be aborted.

- He may flood an accounting layer with newIdentity.
  Flooding an accounting layer with identities is possible. Since the accounting layer is capable of adapting costs to a new identity, it may counter this attack by giving large puzzles to new identities. This affects all new identities and not only those flooding. If a flooding attack is carried out over a long time, a node may decide to split its identity. All recent active users get a new identity, whereas the old one opposes high costs. This would force an attacker to work in intervals and is no longer able to make a permanent DoS attack.

#### 13.9.3.3 Credibility Attack

Another type of DoS attack is the credibility attack. While not a technical attack, it is very effective. A system not having a sufficiently big user base is offering thus a lousy level of anonymity because the anonymity set is too small or the traffic concealing message flow is insufficient.

Another way is to attack the reputation of a system in such a way that the system is no longer used. An adversary has many options to achieve such a reduction in credibility. Examples:

- Disrupt functionality of a system.
  This may be done by blocking of the messaging protocol it uses or by blocking messages. Furthermore, an adversary reduces functionality when removing known participants from the network either by law or by threatening.
- Publicly dispute the effectiveness of a system.
  Disputing the effectiveness is a very effective way to destroy a system. People are not willing to use a system which believed to be compromised if the primary goal of using the system is avoiding being observed.
- Reduce the effectiveness of a system.
  A system may be considerably loaded by an adversary to decrease the positive reception of the system. He may further use the system to send UBM to reduce the overall experience when using the system. Another way of reducing effectiveness is to misuse the system for evil purposes such as blackmailing and making them public.
- Dispute the credibility of the system founders.
  Another way of reducing the credibility of a system is to undermine its creators. If – for example – people believe that a founders' interest was to create a honey pot (e.g., because he is working for a potential state-sponsored adversary) for personal secrets, they will not be willing to use it.
- Dispute the credibility of the infrastructure.
  If the infrastructure is known or suspected to be run by a potential adversary, people's willingness to believe in such a system is expected to be drastically reduced.

# 14 Applied Methodes

Based on the findings of the previous chapter, we used the following methodology in order to find a solution:

1. Identify problem hotspots for a new protocol.
2. Design a protocol that addresses the previously identified hotspots.
3. Build a protocol prototype.
4. Analyse the protocol for weaknesses using attack schemes.
   (a) Tagging/Bugging attacks.
   (b) Tracing attacks.
   (c) Content and identification targeting attacks.
   (d) DoS attacks.

## 14.1 Problem Hotspots

Starting from the previous research, we identified several hotspots that have to be taken care of. The following sections list identified problems and the possible countermeasures which have not been broken in the past.

### 14.1.1 Zero Trust Philosophy

One main disadvantage of almost any system listed in section **??** is that trust (unlimited or limited) has been put into the infrastructure. For example, when using Tor, we need to trust the directory servers. Control over the directory servers might give an attacker the possibility to redirect a connection to controlled entry and exit nodes, which would then break anonymity. In general, control of entry and exit nodes makes a system vulnerable.

To avoid this problem, we decided to apply a zero trust model. We do not trust any platform except for the sending and the receiving computer. We assume that all other devices may be compromised and do create detailed logs about what they are doing. This trust extends partially to our personally known contacts. We believe that some of them might be evil, but they are generally trustworthy. We furthermore assume that traffic on the network layer is observed and recorded at any time. This philosophy creates very hard to meet goals. However, by assuming so, we prevent the system from leaking information through side channels.

**RQ1 (Zero Trust):** *No infrastructure should be trusted unless it is the senders' or the recipients' infrastructure.*

### 14.1.2 Information leakage and P2P Design

An anonymizing system must keep information on messages or their metadata within the system. Ideally, even not disclosing to its members. In a perfectly encrypted system, such metadata is leaked at least by the entry and the exit node. To avoid this, all peers must behave alike. All nodes should be valid endpoints as well as legitimate senders or mixes. Covering all functions in all nodes implies a design with equally built nodes and is shared with many P2P designs.

A fundamental problem of the P2P design is that usually, port forwarding or central infrastructure is required. Technologies such as “hole punching” and “hairpin translation” typically require central infrastructures to support at least the connection and maybe depending on the client infrastructure being used fragile or ineffective. To avoid these problems we decided to rely on traditional centralistic transport infrastructures. As proof of concept, we decided to use SMTP.

The approach supports, however, even mixing transport media. This makes it harder for an attacker to trace a message as the message flow may go through any suitable transport protocol at any time of message transfer.

**RQ2 (Equal nodes):** *Mixes and peers must be indistinguishable from each other.*

To guarantee that information is not leaked through owners of systems or to protect such owners from being forced into cooperation, the system needs to be undetectable.

**RQ3 (Undetectable):** *Nodes should be undistinguishable from regular transport media traffic.*

#### 14.1.2.1 Decoy traffic generation

To create decoy traffic in an untrusted way, we need means to increase and decrease messages in size without knowledge of the routing node. A straightforward approach would be to create decoy traffic in the initial message. Such a design would create a pattern of decreasing or repeating message sizes in the net. To avoid this, we introduced a set of operations to be applied to the original message. The operations are done in such a way that a mixer is unable to tell whether the message size or decrease results in decoy traffic generation/removal or not.

The main message operations are:

- Split and merge messages.
- Add and remove redundancy information.
- Encrypt and Decrypt information.

At this point, we could have used homomorphic encryption instead of redundancy operations. Such encryption would, however, add much complexity to the algorithm with no apparent gain.

#### 14.1.2.2 Message tagging or bugging protection

It is essential to the protocol that any operation at any point of the protocol handling, which is not foreseen, should fail in message transport. This property makes the protocol very fragile, but it prevents mixes from introducing tags which may be followed throughout the system. The protocols counter this fragility by the fact that redundancy added in the message course may be used to recover from misbehaving nodes.

In our approach, we give a single mix called the routing block builder (RBB) full control over the message transport layer. The content used for blending is discardable data. RBB has no control over this aspect. This blending data is ephemeral and will (or may) be removed by the next node. The data received by a mix may be used to generate a “pseudo reply” on the blending layer to transport any other message (related or unrelated) back to the sending node. So tagging on this layer is worthless.

The reason for not giving control over the behavior to this layer to the sender of the message is simple. By giving him control over it, we would allow him to use the information provided here as the primary medium. As an immediate result, the system would be suitable to blackmail any user of the world. It furthermore would create unintentional “exit nodes” to the system, which might oppose further legal threads for participants.

**RQ4 (untagable):** *The message should be un-tagable (neither by a sender nor by an intermediate party such as a mixer).*

**RQ5 (unbugable):** *The message should be unbugable (neither by the sender nor by an intermediate party such as a mixer).*

#### 14.1.2.3 Message replay protection

Message reply protection is crucial for such a system. With the ability to replay a message, an adversary may “highlight” a message flow as it would always generate the same traffic pattern. So there needs to be a reply pattern protecting the protocol from message replay. As we do have MURBs in our protocol, this is a problem. A MURB is by design replayable. We, therefore, need a possibility for the original sender using a MURB to make messages distinguishable, which may not be used by an adversary.

**RQ6 (replay):** *A message must not be replayable.*

It should be able to increase and shrink in size, or all messages must have a uniform size. Decoy traffic should not be distinguishable from message traffic.

#### 14.1.2.4 No Dedicated Infrastructure Philosophy

There should be no infrastructure dedicated to the operation of the solution. This avoids a single point of failure, as well as the possibility for an adversary to shut down this infrastructure to disrupt the functioning of the system as a whole. This requirement is already covered implicitly in RQ1 (Zero Trust).

### 14.1.3 Accounting

The infrastructure must not be misused as UBM sending infrastructure. This implies that sending messages is connected to some "cost". "Costs" must be connected to some identity to allow accounting. Linking to a global identity would allow assigning traffic to a real-world user. Therefore the protocol must allow creating ephemeral local identities not linked to a real identity.

**RQ7 (accounting):** *The system must be able to do accounting without being linked to a real identity.*

### 14.1.4 Anonymisation

The system must allow the anonymizing of message source and message destination at any point. It should not be visible to the infrastructure protocol whether a message has reached its destination or not.

**RQ8 (anonymisation):** *A system must be able to anonymize sender and recipient at any point of the transport layer and any point of mixing unless it is the sender or the recipient itself.*

### 14.1.5 Initial Bootstraping

The system must allow bootstrapping from a zero-knowledge or near-zero knowledge point. Therefore, the protocol must be able to extend the network of known nodes on its own.

**RQ9 (bootstrapping):** *The system must allow to bootstrap from a zero-knowledge or near-zero-knowledge point and extend the network on its own.*

### 14.1.6 Cypher selection

In this protocol, a lot of encryption and hashing algorithms have to be used. This choice of these algorithms should be explained.

From the requirements side, we have to follow the following principle:

**RQ10 (algorithmic variety):** *The system must be able to use multiple symmetric, asymmetric, and hashing algorithms to immediately fall back to a secure algorithm for all new messages if required.*

First of all, we need a subset of encryption algorithms all implementations may rely on. Defining such a subset guarantees interoperability between all nodes regardless of their origins.

Secondly, we need to have a spectrum of algorithm in such a manner that it may be (a) enlarged if necessary and (b) there is an alternative if an algorithm (or a mathematical problem class) is broken (so that algorithms may be withdrawn if required without affecting the function in general).

And third, due to the onion-like design described in this document, asymmetric encryption should be avoided in favor of symmetric encryption to minimize losses due to the key length and the generally higher CPU load opposed by asymmetric keys.

If the algorithm is generally bound to specific key sizes (due to S-Boxes or similar constructs), the key size is incorporated into the definition. If not, the key size is handled as a parameter.

The key sizes have been chosen in such a manner that the key types form tuples of approximately equal strength. The support of Camelia192 and Aes192 has been defined as optional. However, as they are wildly common in implementations, they have already been standardized as they build a possibility to step up security in the future.

Having these criteria for choice, we chose to use the following keys and key sizes:

- Symmetric
  - AES (key sizes: 128, 192, 256)
  - Camellia (key sizes: 128, 192, and 256)
- Asymmetric
  - RSA (key size: 2048, 4096, and 8192)
  - Named Elliptic Curves

    - secp384r1
    - sect409k1
    - secp521r1

- Hashing
  - sha3-256
  - sha3-384
  - sha3-512
  - RIPE-MD160
  - RIPE-MD256
  - RIPE-MD320

Within the implementation, we assigned algorithms to a security strength level:

- LOW
  AES128, Camellia128, RSA1024, sha3-256
- MEDIUM
  AES192, Camellia 192, RSA2048, ECC secp384r1, sha3-256
- HIGH
  AES256, Camellia256, RSA4096, ECC sect409k1, sha3-384
- QUANTUM
  AES256, Camellia256, RSA8192, ECC secp521r1, ntru, sha3-512

This allows categorizing the used algorithms to a strength. This list, however, should only serve the purpose of selecting algorithms for people without cryptological know-how.

### 14.1.7 Reed-Solomon function

Originally [**reed1960polynomial**] introduced a system allowing the use of polynomial codes to create error-correcting codes. In [**chaum1988multiparty**] **chaum1988multiparty**, they have shown that the codes are suitable for distributing data assuming enough parties are honest.

Unlike Chaum et al.'s proposition, we are not using the Reed Solomon function to achieve anonymity or privacy. Instead, we use it for decoy traffic generation. We are splitting a message into multiple parts at several points when routing and assemble it again on different nodes. By doing so, we achieve two vital things. First, we introduce the possibility of recovering errors due to misbehaving nodes, and secondly, the real traffic can no longer be differentiated from decoy traffic.

### 14.1.8 Usability

The system must be usable without cryptographic know-how and with popular tools. This is necessary to accept the system broadly and makes it easy to use for peoples already communicating.

**RQ11 (easy handleable):** *The system must be usable without cryptographic know-how and with popular tools.*

## 14.2 Protocol outline

The protocol itself is independent of the transport layer specified. We emphasize in this section to the general building blocks, the cryptographic structure, and the general protocol attributes. In section **??**, we will then further elaborate on the protocols' inner structure.

The protocol is built on multiple layers. On the logic side, the protocol is split into two parts:

1. Transport Layer
   Standard Internet infrastructures provide this Layer. The primary goal is to hide or blend our protocol into regular traffic within that layer.

2. Blending and subsequent layers
   Any user of the Internet may provide these layers. Since these layers may be mixes-only, or valid endpoints. Mixes may or may not be publicly known. In a first implementation, we build this system as a standard Java application. The primary goal is to compile it to native code afterward and run it on an SoC like infrastructure such as a RaspberryPi or port it to an android device.

   We may further split these layers into

   (a) Blending layer
   This layer takes messages and creates transport layer conformant messages. In an ideal case, the messages generated by this layer are indistinguishable from the regular message traffic, and the embedded message is only visible for the receiving node.

(b) Routing layer
The routing layer disassembles and reassembles messages. This operation guarantees that messages are generated in such a way that decoy traffic is not differentiable from non-decoy traffic.

(c) Accounting layer
The accounting layer has three jobs. First, he has to authorize the message processing after decrypting the header block. Secondly, he handles all header request blocks and the reply blocks. And third, it keeps track of the accounting regarding the sent messages.

### 14.2.1 Protocol Terminology

For our protocol, we use the following terms:

- **Sender:** The user or process originally composing the message.
- **Recipient:** The user or process destined to receive the message in the end.
- **Router:** Any node which is processing the message. Please note that all nodes are routers.
- **Message:** The “real content” to be transferred from the sender to the recipient.
- **Payload:** Any data transported between routers regardless of the meaningfulness or relevance to the message.
- **Decoy traffic:** Any data transported between routers that have no relevance to the message at the final destination.
- **Identity:** A tuple of a routable address, and a public key. This tuple is a long-living tuple but may be exchanged from time to time.
- **Ephemeral Identity:** An identity created on any node with a limited lifetime anyone possessing the private key (proven by encrypting with it) is accepted as representative of that identity.
- **Routing Block Builder (RBB):** An entity, which is building a routing block. Typically identical to either sender or receiver.

### 14.2.2 Vortex Communication model

In this section, we introduce a new consistent, transport-independent model for representing the different protocols used by MessageVortex.

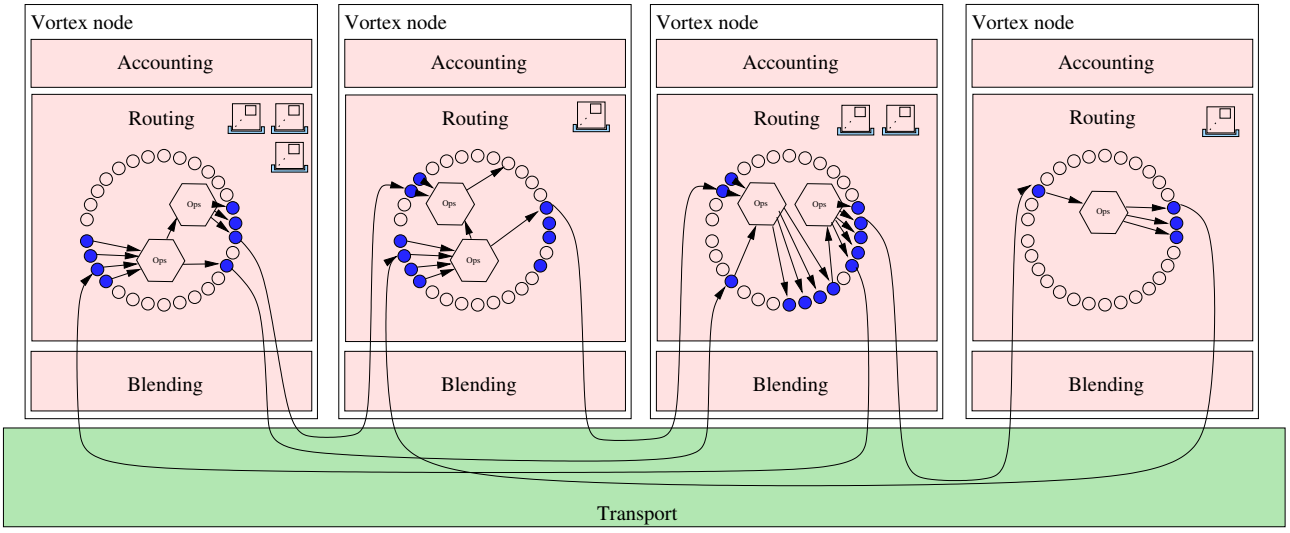


Figure 14.1: A rough protocol outline of the MessageVortex protocol

We divide our protocol into four different layers, whereas only three are specific to the MessageVortex protocol. The lowest layer is the transport layer. As expressed earlier, dedicated protocols are easy to censor. Therefore we build our protocol on top of other suitable transport protocols.

The other Three layers are vortex specific and do not require any infrastructure on the Internet. We elaborate further on these layers in the next section.

### 14.2.3 Transport Layer

For our first tests, we used a custom transport layer, allowing us to monitor all traffic quickly, and build structures in a very flexible way. This transport layer works locally with a minimum amount of work for setup and deployment. It furthermore works across multiple hosts in a broadcast domain. The API may be used to support almost any kind of transport layer.

After that, we focussed on the protocols identified in the previous sections for transport:

- SMTP
- XMPP

For the prototype, we have implemented an SMTP transport agent and the respective blending layer.

### 14.2.4 Blending Layer

The blending layer is taking care of multiple problems:

- It is translating the message block into a suitable format for transport
  This translation includes jobs such as embedding a block as encoded text, as a binary attachment or hide it within a message using steganography.
- Extract incoming blocks
  Identify incoming messages containing a possible block and extract it from the message.
- Do housekeeping on the storage layer of the transport protocol
  Access protocols POP and IMAP require that messages are deleted from time to time to stay below the sizing quotas of an account.

We define the blending layer to work as follows when receiving messages:

1. Log arrival time (in UTC) on the transport layer.
2. Extract possible blocks.
3. Apply decryption on a suspected header block.
4. Identify the header block as valid by querying the accounting.
5. Extract and decrypt subsequent blocks.
6. Pass extracted blocks and information to the routing layer.

We define the blending layer to work as follows for sending messages:

1. Assemble message as passed on by the routing layer.
2. Using the blending method specified in the routing block, build an empty message.
3. Create a message decoy content.
4. Send the message to the appropriate recipient using the transport layer protocol.

There is no specification on the housekeeping part of the blending layer, as this part is specific to the requirements of the account owner. We do, however, recommend to handle messages precisely as if the messages would be handled on an account handled by a human.

### 14.2.5 Routing Layer

The routing layer receives the message blocks in a decrypted and authorized form from the blending layer and processes them as follows:

- Build structure representing the block building and the appropriate block IDs.
- Schedule all Routing blocks for processing in a priority queue.
- Authorise all routing blocks ready for processing with the calculated block sizes.
- Process blocks.
- Send prepared building blocks to the Blending layer.

#### 14.2.5.1 Block Structure

A VortexMessages' main block structure is a sequence of blocks. This block sequence starts with a header containing a symmetric key encrypted with the public key of the current node and a header block containing the immediate details to decrypt the subsequent blocks (if any).

A routing block follows the header block. This routing block contains the information required for subsequent routing. According to the instructions in this block, valid data blocks may be processed, assembled, and sent to a subsequent location.

The next block is the routing log block. This block protocols the routing information of a message and is somewhat similar to an onionized variant of the received headers in SMTP.

The last part of the message is a sequence of data blocks. They contain the actual data or decoy traffic.

#### 14.2.5.2 MURBs

The protocol includes the capability of MURBs. Such MURBs enable a user to send a limited amount of times messages to an anonymous receiver. Such sending may be done without having any knowledge about its identity, the location, or infrastructure he is using.

A MURB in our term is an entirely prepared routing instruction built by the recipient of a message. The sender has only the routing blocks and the instructions to assemble the initial message. It does not know the message path except for the first message hops.

As a MURB is a routing block, it generates the same pattern on the network each time a sender uses it. To avoid statistical visibility, we need to limit the number of uses per MURB. As a maximum number of usages, the protocol is limited to 127 usages. This number should be sufficiently sized for automated messages. A minute pattern would disappear after 2 hours latest and an hourly pattern after five days.

For a MURB to work, the RBB has to take care that all quotas required to the route are sufficiently sized. Such sizing is hard to foresee in some cases. An RBB may query these identities from time to time to make sure that they do not deplete. Wherever possible, MURBs should be dropped in favor of multiple SURBs to avoid the dangers of MURBs.

## 14.3 Protocol handling

In the following sections, we outline the handling of messages we split the handling into incoming messages and outgoing messages. All handling assumes that we have a blending layer independently picking up messages as advertised in the capabilities messages.

### 14.3.1 Block Processing

A Block is picked up in the blending layer and then handled in the routing layer. First, we try to authenticate the message. If we can authenticate the message, we process it and add the contained instructions to a processing workspace. Unauthenticated messages may be discarded at any point.

The processing of a sending block is triggered by a routing block in the workspace, as shown in figure 14.2. The assembly instructions are processed to collect the payload blocks. Then the encryption is applied to the message and passed on to the blending layer for processing.

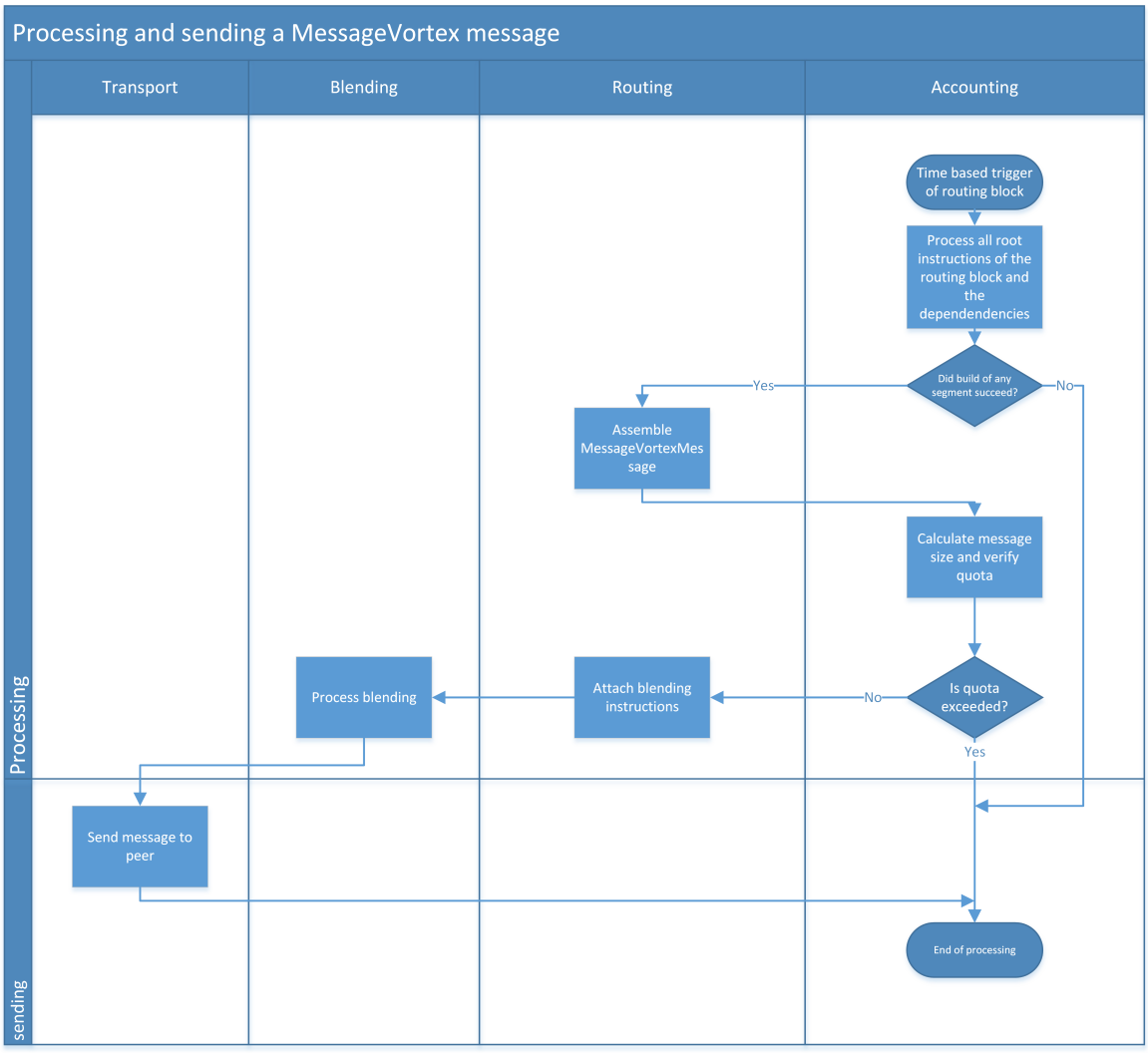


Figure 14.2: flow diagram showing processing of outgoing messages

## 14.4 Sub Research Questions Roundup

We sum up the findings of the last part regarding our three research questions and describe the next steps to be taken.

### 14.4.1 SQ1: Technologies for sending messages maintaining unlinkability against an adversary

We were unable to identify a single technology that withstands our adversary model entirely. The technologies were either too simple to withstand an adversary (e.g., remailers), have substantial flaws affecting their reliability (e.g., most mixes and DC networks), an active adversary could sabotage them or do not scale.

We were able to describe a rough protocol that performs far better in almost all aspects of anonymity than the solutions described in the previous sections. This comparison was always made for the adversary model given in section 16.1. If we assume that the constraints of trust (only trust in sender and recipient infrastructure, whereas we always have multiple recipients) are valid, we can make the following statements regarding anonymity and unlinkability:

- If an adversary identifies all involved nodes of a message and identifies all the corresponding messages and controls, all nodes except for senders and recipients nodes, he can determine message frequency, maximum message size, and message peers.
- If an adversary can identify all involved $i$ nodes of a sending party while controlling $j$ nodes, then he may determine a $k$-anonymity set whereas $k = i - j$ for the message set and a maximum message frequency.
- If an adversary is running a node, he may identify other nodes participating in the network by analyzing peer messages.

We may safely assume that a carefully crafted message within a standard message flow is therefore unlinked from the two message peers. An adversary running a node may identify over time, possibly participating nodes, if not operating in a closed group, but he will be unable to query or use such a node without the corresponding keys. He may be able to observe such nodes, possibly view their activity, but he is unable to match messages generated.

In the next section, we will further elaborate on this protocol and analyze it. We will focus on the question of whether it is possible to create a protocol that withstands our threat model.

### 14.4.2 SQ2: Attacking unlinkability and circumvention

In the previous part, we identified a lot of attack schemes used to attack the anonymity of a protocol or infrastructure. While not all are technical, technicality plays a major part. We identified:

- Anonymity attacks
  - Hotspot attacks
  - Side-channel attacks
  - Sizing analysis
  - Bugging attacks
  - Tagging and tracing attacks
    - Peer discovering attacks
    - Traffic flow attacks
- Availability and reliability attacks
  - Denial of service (DoS) attacks
  - Censorship attacks
- Non-technical Attacks
  - Credibility attacks
  - Censorship attacks

We identified several possibilities to circumvent the attack classes listed above. Some of them, such as the bugging attack, may be countered by design (e.g., by allowing only simple messages). Others can only be countered partially in reality (e.g., DDoS attacks). We will further elaborate on the protocol and then analyze the impact of every single attack on the protocol.

### 14.4.3 SQ3: Attack Mitigation by design

This SQ is a part of the previous question to a certain extent. We identified that reliability and trust are key factors to a protocol. Therefore, allowing a single point of failure (SPOF) or extending trust over central infrastructures is deadly for an anonymizing protocol. Undetectability is another crucial point ignored by almost all protocols except for some aspects of Tor and some advanced forms of remailers.

When elaborating on the protocol in the next part, we will focus on introducing designs that will prohibit actions endangering anonymity. In the Result section, we will focus on all attacks, which should be mitigated by design.

# Part IV

# The MessageVortex System

In the following section we describe the core functionalities

MessageVortex is a protocol piggybacking non transport protocols somehow similar as S/MIME[**RFC2015**] or PGP[**PGP**] which are common transport protocols such as SMTP.

# 15 Intro

# 16 Requirements for an Anonymizing Protocol

In the following sections, we elaborate on the main characteristics of the anonymizing protocol.

The primary goal of the protocol is to enable Freedom of speech, as defined in Article 19 of the International Covenant on Civil and Political Rights (ICCPR)[**iccpr**].

> everyone shall have the right to hold opinions without interference

and

> Everyone shall have the right to freedom of expression; this right shall include freedom to seek, receive and impart information and ideas of all kinds, regardless of frontiers, either orally, in writing or print, in the form of art, or through any other media of his choice.

We imply that not all participants on the Internet share this value. As of September þ1, 2016 Countries such as China (signatory), Cuba (signatory), Qatar, Saudi Arabia, Singapore, United Arab Emirates, or Myanmar did not ratify the ICCPR. Other countries such as the United States or Russia did either put local laws in place superseding the ICCPR or made reservations rendering parts of it ineffective. We may, therefore, safely assume that freedom of speech is not given on the Internet, as at least countries explicitly supersede them.

Network packets may pass through any point of the world. A sender has no control over it. This lack of control is since every routing device decides on its own for the next hop. This decision may be based on static rules or influenced by third party nodes or circumstances (e.g., BGB, RIP, OSPF...). It is furthermore not possible to detect what way has a packet taken. The standard network diagnostic tool `tracert` respectively `traceroute` returns a potential list of hops. This list is only correct under certain circumstances (e.g., a stable route for multiple packets or same routing decisions regardless of other properties than the source and destination address). Any Output of these tools may, therefore, not taken as a log of routing decisions. There is no possibility in standard IP routed networks to foresee a route for a packet, nor can it be measured, recorded, or predicted before, while, or after sending.

As an example of the problems analyzing a packet route, we may look at `traceroute`. According to the man page of traceroute, `traceroute` uses UDP, TCP, or ICMP packets with a short TTL and analyses the IP of the peer sending a TIME_EXCEEDED (message of the ICMP protocol). This information is then collected and shown as a route. This route may be completely wrong. The man page describes some of the possible causes.

We cannot state that data packets we are sending are passing only through countries accepting the ICCPR to the full extent, nor can we craft packages following such a rule.

```
$traceroute www.ietf.org
traceroute to www.ietf.org.cdn.cloudflare-dnssec.net (104.20.0.85), 64 hops max
1 147.86.8.253 0.418ms 0.593ms 0.421ms
2 10.19.0.253 1.177ms 0.829ms 0.782ms
3 10.19.0.253 0.620ms 0.427ms 0.402ms
4 193.73.125.35 1.121ms 0.828ms 0.905ms
5 193.73.125.81 2.991ms 2.450ms 2.414ms
6 193.73.125.81 2.264ms 1.961ms 1.959ms
7 192.43.192.196 6.472ms 199.543ms 201.152ms
8 130.59.37.105 3.465ms 3.138ms 3.121ms
9 130.59.36.34 3.904ms 3.897ms 4.989ms
10 130.59.38.110 3.625ms 3.333ms 3.379ms
11 130.59.36.93 7.518ms 7.232ms 7.246ms
12 130.59.38.82 7.155ms 17.166ms 7.034ms
13 80.249.211.140 22.749ms 22.415ms 22.467ms
14 104.20.0.85 22.398ms 22.222ms 22.146ms
```

Figure 16.1: A traceroute to the host www.ietf.org

To enable freedom of speech, we need a mean of transport for messages which keep sender and recipient anonymous.

## 16.1 Threat model

We refer to jurisdiction as a geographical area where a set of legal rules created by a single actor or a group of actors apply, which contains executive capabilities (e.g., police, army, or secret service) to enforce this set of legal rules.

We assume for our protocol that adversaries are state-sponsored actors or players of large organizations. These actors have high funding and expected to have elaborated capabilities themselves or within reach of the sponsor. Actors may join forces with other actors as allies. However, achieving more than 50% on a world scale is excluded from our model. We always assume one or more actors with disjoint interests covering half of the network or more.

We assume the following goals for an adversary:

- An adversary may want to disrupt non-authorized communication.

- An adversary may want to read any information passing through portions of the Internet.
- An adversary may want to build and conserve information about individuals or groups of individuals of any aspect of their life.

To achieve these goals, we assume the following properties of our adversary:

- An adversary has elaborated technical know-how to attack any infrastructure. This attack may cover any attack favoring his goals, starting with exploiting weaknesses of popular software (e.g., buffer overflows or zero-day exploits) down to simple or elaborated (D)DoS attacks.
- An adversary may monitor traffic at any point in public networks within a jurisdiction.
- An adversary may modify routing information within a jurisdiction freely.
- An adversary may freely modify even cryptographically weak secured data where a single or a limited number of entities grant proof of authenticity or privacy.
- An adversary may inject or modify any data on the network of a jurisdiction.
- An adversary may create their nodes in a network. He may furthermore monitor their behavior and data flow without limitation.
- An adversary may force a limited number of other non-allied nodes to expose their data to him. For this assumption, we explicitly excluded actors with disjoint interests.
- An adversary may have similar access to resources as within its jurisdiction in a limited number of other jurisdictions.

we may furthermore subdivide the adversaries into the following sub-classes:

- A censoring adversary
  The primary goal of this adversary is censoring messages and opinions, not within his interests. He does this, regardless of whether the activities of censorship may be observed or not. Therefore, this adversary does not cloak its activities and typically bans censorship circumventing activities as illegal.
- An observing adversary
  This adversary behaves like a traditional spy. He collects and classifies information while hiding its activities. Unlike within reach of a censoring adversary, in this case, typically, no restrictions apply to the use of anonymization technology.

## 16.2 Required Properties of an unobservable network

In this section, we summarize the required properties of an anonymizing system.

### 16.2.1 Anonymizing and Unlinking

As we are unable to limit the route of our packets through named jurisdictions, we must protect ourselves from unintentionally breaking the law of a foreign country. Therefore, we need to be anonymous when sending or receiving messages. Unfortunately, most transport protocols (in fact, almost all of them such as SMTP, SMS, XMPP, or IP) use a globally unique identifier for senders and receivers, which are readable by any party which is capable of reading the packets.

As a result, the anonymization of a sender or a receiver is not simple. A relay may allow at least the anonymization of the original sender given trust into the proxy. By combining it with encryption, we may even achieve a simple form of a sender and receiver pseudonymity. If cascading more relay like infrastructures and combining it with cryptography, we may achieve sender and receiver anonymity. When introducing anonymous remailing endpoints, we may additionally achieve both simultaneously.

These are the standard approaches in remailers and mixes. Their approaches are questionable as shown in **??** and **??**. We have seen attacks on such systems in the past. Some of them were successful.

### 16.2.2 Censorship Resistant

In our scenario in 16, we defined the adversary as someone with superior access to the network and its infrastructure. Such an adversary might attack a message flow in several ways:

- Identify sender
- Identify recipient
- Read messages passed or extract meta information
- Disrupt communication fully or partially

We furthermore have to assume that all actions taken by a potential adversary are not subject to legal prosecution. This assumption based on the fact that an adversary trying to establish censorship may be part of the government of jurisdiction. We may safely assume that there are legal exceptions in some jurisdictions for such entities.

To be able to withstand an adversary outlined above, the messages sent requires to be unidentifiable by attributes or content. “Attributes” include any meta information including, but not limited to, frequency, timing, message size, sender, protocol, ports, or recipient.

### 16.2.3 Controllable trust

We have multiple options for relying on trust when building our system. We may rely on trust in infrastructure, we may work with distrust in infrastructure. In our model, we will work with distrust into the infrastructure. As every infrastructure node learns from each transaction (e.g., the usage of the network or size of messages), we have to minimize or ideally eradicate such information gains. A main problem is that we are unable to hide peer senders or recipients when routing messages. In jurisdictions where such infrastructure usage is illegal, we need to hide the presence of our routing messages from any party not trusted. Such hiding concludes that we need to be able to control which nodes are involved when sending messages. We refer to this concept as controllable trust.

In terms of trust, we have to conclude that:

1. We trust in infrastructure because it is under full control of either the sender or the recipient.
2. We should not trust all other infrastructure as an adversary is potentially able to misuse data passed through it.

In this work, we work with both cases. We will, however, avoid whenever possible to trust in any third party apart from the sender and recipient.

### 16.2.4 Reliable

Any message-sending protocol needs to be reliable in its functionality. If the means of message transport are unreliable, users tend to use different means for communication[**zhou2011examining**].

### 16.2.5 Diagnoseable

Transparent behavior is a prerequisite for reliability. If something is generating a behavior, but we are unable to determine the reason for it (i.e., if we are expecting a different behavior), we usually assume a malfunction. Therefore “reliable” means not only stable by its behavior. It also means diagnoseable. A user’s perception will not be “reliable” if he is not able to determine causes for differences in observed and expected behavior (e.g., [**nicholson2003assessing**]).

### 16.2.6 Available

Availability has two meanings in this context, which do differ. Technology is available if. . .

1. a sender and a recipient have (or may have) the means of using it.
2. the infrastructure provides the service (as opposed to: “is running in a degraded or faulty state and, therefore, unable to provide the service”).

The first meaning tells us that a protocol must run on infrastructure on which the user has access to it.

The second meaning tells us that messages must always be capable of flowing from the sender to the recipient. As a part of the infrastructure may fail at any time, the protocol must offer the possibility to send messages through alternate routes. Alternative routes are simple to achieve, and many protocols implement such redundancies already. However, taking into account that the sender and recipient are not known to a routing node, this is a goal hard to achieve. If we leave the choice of routing to any node apart from a trusted node, we will enable untrusted nodes to manipulate routing decisions and thus affect the security of a message.

### 16.2.7 Identifiable Sender

A messaging system offering unlinkability may offer sender anonymity. If so, a sender should be identifiable in such a way, that a classification of senders is possible at any time, and impersonation is not achievable. It is important to understand that an identifiable sender does not necessarily mean that we can identify a sender as a specific party. In our case, any identification will do, which offers non-hijackable pseudonymity. We decided to go for a short-lived pseudonymity (see eID in section **??**). This system guarantees that while only a pseudonym of the sender is known, the hijacking of data by other participants of the system is not possible.

## 17 Rationale

## 18 Protocol Outline

## 19 Key Components

## 19.1 Nodes

# Part V

# Implementation

# Part VI

# Operational concerns

## 25 Routing

### 25.1 Algorithms Suitable for Achieving Anonymity

### 25.2 Possibilities of Routing Diagnosis and Reputation Building

### 25.3 Possibilities of Redundancies

## 26 Protocol Bootstrapping

### 26.1 Key Distribution for Endpoints

### 26.2 Key Aquisition for Routing Nodes

# Part VII

# Analysis of MessageVortex

# 27 Identification of Possible Attack Schemes and Mitigation

## 27.1 Static Attacks

### 27.1.1 Bugging and Tagging Attacks

### 27.1.2 Information Leaking related to Information Available to Routing Nodes

### 27.1.3 Identification of involved Nodes

### 27.1.4 Identification of MessageVortex Traffic

## 27.2 Dynamic Attacks

### 27.2.1 Attacks against the vortex system itself

### 27.2.2 Attacking a single ephemeral Identity of a MessageVortex Node

### 27.2.3 Attacking Sending and Receiving Identities of the MessageVortex System

### 27.2.4 Recovery of Previously Carried Out Operations

# 28 Analysis of the effectiveness of Attack Schemes

# 29 Analysis of the Degree of Anonymization in Comparison to other Systems

# Part VIII

# Discussion on Results

# Part IX

# Appendix

# A The RFC draft document

Workgroup: Internet Engineering Task Force
Internet-Draft: draft-gwerder-messagevortexmain-04
Published: 23 November 2019
Intended Status: Experimental
Expires: 26 May 2020
Author: M. Gwerder
*FHNW*

## MessageVortex Protocol

### Abstract

The MessageVortex (referred to as Vortex) protocol achieves different degrees of anonymity, including sender, receiver, and third-party anonymity, by specifying messages embedded within existing transfer protocols, such as SMTP or XMPP, sent via peer nodes to one or more recipients.

The protocol outperforms others by decoupling the transport from the final transmitter and receiver. No trust is placed into any infrastructure except for that of the sending and receiving parties of the message. The creator of the routing block has full control over the message flow. Routing nodes gain no non-obvious knowledge about the messages even when collaborating. While third-party anonymity is always achieved, the protocol also allows for either sender or receiver anonymity.

### Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 26 May 2020.

## Copyright Notice

# Table of Contents

# 1. Introduction

Anonymisation is hard to achieve. Most previous attempts relied on either trust in a dedicated infrastructure or a specialized networking protocol.

Instead of defining a transport layer, Vortex piggybacks on other transport protocols. A blending layer embeds Vortex messages (VortexMessage) into ordinary messages of the respective transport protocol. This layer picks up the messages, passes them to a routing layer, which applies local operations to the messages, and resends the new message chunks to the next recipients.

A processing node learns as little as possible from the message or the network utilized. The operations have been designed to be sensible in any context. The 'onionized' structure of the protocol makes it impossible to follow the trace of a message without having control over the processing node.

MessageVortex is a protocol which allows sending and receiving messages by using a routing block instead of a destination address. With this approach, the sender has full control over all parameters of the message flow.

A message is split and reassembled during transmission. Chunks of the message may carry redundant information to avoid service interruptions during transit. Decoy and message traffic are not differentiable as the nature of the addRedundancy operation allows each generated portion to be either message or decoy. Therefore, any routing node is unable to distinguish between message and decoy traffic.

After processing, a potential receiver node knows if the message is destined for it (by creating a chunk with ID 0) or other nodes. Due to missing keys, no other node may perform this processing.

This RFC begins with general terminology (see Section 2) followed by an overview of the process (see Section 3). The subsequent sections describe the details of the protocol.

## 1.1. Requirements Language

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 [RFC2119].

### 1.2. Protocol Specification

Appendix A specifies all relevant parts of the protocol in ASN.1 (see [CCITT.X680.2002] and [CCITT.X208.1988]). The blocks are DER encoded, if not otherwise specified.

### 1.3. Number Specification

All numbers within this document are, if not suffixed, decimal numbers. Numbers suffixed with a small letter 'h' followed by two hexadecimal digits are octets written in hexadecimal. For example, a blank ASCII character (' ') is written as 20h and a capital 'K' in ASCII as 4Bh.

## 2. Entities Overview

The following entities used in this document are defined below.

### 2.1. Node

The term 'node' describes any computer system connected to other nodes, which support the MessageVortex Protocol. A 'node address' is typically an email address, an XMPP address or other transport protocol identity supporting the MessageVortex protocol. Any address SHOULD include a public part of an 'identity key' to allow messages to transmit safely. One or more addresses MAY belong to the same node.

#### 2.1.1. Blocks

A 'block' represents an ASN.1 sequence in a transmitted message. We embed messages in the transport protocol, and these messages may be of any size.

#### 2.1.2. NodeSpec

A nodeSpec block, as specified in Section a.6, expresses an addressable node in a unified format. The nodeSpec contains a reference to the routing protocol, the routing address within this protocol, and the keys required for addressing the node. This RFC specifies transport layers for XMPP and SMTP. Additional transport layers will require an extension to this RFC.

#### 2.1.2.1. NodeSpec for SMTP nodes

An alternative address representation is defined that allows a standard email client to address a Vortex node. A node SHOULD support the smtpAlternateSpec (its specification is noted in ABNF as in [RFC5234]). For applications with QR code support, an implementation SHOULD use the smtpUrl representation.

```
localPart         = <local part of address>
domain            = <domain part of address>
email             = localPart "@" domain
keySpec           = <BASE64 encoded AsymmetricKey [DER encoded]>
smtpAlternateSpec = localPart ".." keySpec ".." domain "@localhost"
smtpUrl           = "vortexsmtp://" smtpAlternateSpec
```

This representation does not support quoted local part SMTP addresses.

#### 2.1.2.2. NodeSpec for XMPP nodes

Typically, a node specification follows the ASN.1 block NodeSpec. For support of XMPP clients, an implementation SHOULD support the jidAlternateSpec (its specification is noted in ABNF as in [RFC5234]).

```
localPart         = <local part of address>
domain            = <domain part of address>
resourcePart      = <resource part of the address>
jid               = localPart "@" domain [ "/" resourcePart ]
keySpec           = <BASE64 encoded AsymmetricKey [DER encoded]>;
jidAlternateSpec  = localPart ".." keySpec ".."
                    domain "@localhost" [ "/" resourcePart ]
jidUrl            = "vortexxmpp://" jidAlternateSpec
```

## 2.2. Peer Partners

This document refers to two or more message sending or receiving entities as peer partners. One partner sends a message, and all others receive one or more messages. Peer partners are message specific, and each partner always connects directly to a node.

## 2.3. Encryption keys

Several keys are required for a Vortex message. For identities and ephemeral identities (see below), we use asymmetric keys, while symmetric keys are used for message encryption.

### 2.3.1. Identity Keys

Every participant of the network includes an asymmetric key, which SHOULD be either an EC key with a minimum length of 384 bits or an RSA key with a minimum length of 2048 bits.

The public key must be known by all parties writing to or through the node.

### 2.3.2. Peer Key

Peer keys are symmetrical keys transmitted with a Vortex message and are always known to the node sending the message, the node receiving the message, and the creator of the routing block.

A peer key is included in the Vortex message as well as the building instructions for subsequent Vortex messages (see RoutingCombo in Appendix A).

### 2.3.3. Sender Key

The sender key is a symmetrical key protecting the identity and routing block of a Vortex message. It is encrypted with the receiving peer key and prefixed to the identity block. This key further decouples the identity and processing information from the previous key.

A sender key is known to only one peer of a Vortex message and the creator of the routing block.

## 2.4. Vortex Message

The term 'Vortex message' represents a single transmission between two routing layers. A message adapted to the transport layer by the blending layer is called a 'blended Vortex message' (see Section 3).

A complete Vortex message contains the following items:

- The peer key, which is encrypted with the host key of the node and stored in a prefixBlock, protects the inner Vortex message (innerMessageBlock).
- The small padding guarantees that a replayed routing block with different content does not look the same.
- The sender key, also encrypted with the host key of the node, protects the identity and routing block.
- The identity block, protected by the sender key, contains information about the ephemeral identity of the sender, replay protection information, header requests (optional), and a requirement reply (optional).
- The routing block, protected by the sender key, contains information on how subsequent messages are processed, assembled, and blended.
- The payload block, protected by the peer key, contains payload chunks for processing.

## 2.5. Message

A message is content to be transmitted from a single sender to a recipient. The sender uses a routing block either built itself or provided by the receiver to perform the transmission. While a message may be anonymous, there are different degrees of anonymity as described by the following.

- If the sender of a message is not known to anyone else except the sender, then this degree is referred to as 'sender anonymity.'
- If the receiver of a message is not known to anyone else except the receiver, then the degree is 'receiver anonymity.'
- If an attacker is unable to determine the content, original sender, and final receiver, then the degree is considered 'third-party anonymity.'
- If a sender or a receiver may be determined as one of a set of <k> entities, then it is referred to as k-anonymity[KAnon].

A message is always MIME encoded as specified in [RFC2045].

## 2.6. Key and MAC specifications and usage

MessageVortex uses a unique encoding for keys. This encoding is designed to be small and flexible while maintaining a specific base structure.

The following key structures are available:

- SymmetricKey
- AsymmetricKey

MAC does not require a complete structure containing specs and values, and only a MacAlgorithmSpec is available. The following sections outline the constraints for specifying parameters of these structures where a node MUST NOT specify any parameter more than once.

If a crypto mode is specified requiring an IV, then a node MUST provide the IV when specifying the key.

### 2.6.1. Asymmetric Keys

Nodes use asymmetric keys for identifying peer nodes (i.e., identities) and encrypting symmetric keys (for subsequent de-/encryption of the payload or blocks). All asymmetric keys MUST contain a key type specifying a strictly-normed key. Also, they MUST contain a public part of the key encoded as an X.509 container and a private key specified in PKCS#8 wherever possible.

RSA and EC keys MUST contain a keySize parameter. All asymmetric keys SHOULD contain a padding parameter, and a node SHOULD assume PKCS#1 if no padding is specified.

NTRU specification MUST provide the parameters "n", "p", and "q".

### 2.6.2. Symmetric Keys

Nodes use symmetric keys for encrypting payloads and control blocks. These symmetric keys MUST contain a key type specifying a key, which MUST be in an encoded form.

A node MUST provide a keySize parameter if the key (or, equivalently, the block) size is not standardized or encoded in the name. All symmetric key specifications MUST contain a mode and padding parameter. A node MAY list multiple padding or mode parameters in a ReplyCapability block to offer the recipient a free choice.

## 2.7. Transport Address

The term 'transport address' represents the token required to address the next immediate node on the transport layer. An email transport layer would have SMTP addresses, such as 'vortex@example.com,' as the transport address.

## 2.8. Identity

### 2.8.1. Peer Identity

The peer identity may contain the following information of a peer partner:

- A transport address (always) and the public key of this identity, given there is no recipient anonymity.
- A routing block, which may be used to contact the sender. If striving for recipient anonymity, then this block is required.
- The private key, which is only known by the owner of the identity.

### 2.8.2. Ephemeral Identity

Ephemeral identities are temporary identities created on a single node. These identities MUST NOT relate to another identity on any other node so that they allow bookkeeping for a node. Each ephemeral identity has a workspace assigned, and may also have the following items assigned.

- An asymmetric key pair to represent the identity.
- A validity time of the identity.

### 2.8.3. Official Identity

An official identity may have the following items assigned.

- Routing blocks used to reply to the node.
- A list of assigned ephemeral identities on all other nodes and their projected quotas.
- A list of known nodes with the respective node identity.

### 2.9. Workspace

Every official or ephemeral identity has a workspace, which consists of the following elements.

- Zero or more routing blocks to be processed.
- Slots for a payload block sequentially numbered. Every slot:
  - MUST contain a numerical ID identifying the slot.
  - MAY contain payload content.
  - If a block contains a payload, then it MUST contain a validity period.

### 2.10. Multi-use Reply Blocks

'Multi-use reply blocks' (MURB) are a special type routing block sent to a receiver of a message or request. A sender may use such a block one or several times to reply to the sender linked to the ephemeral identity, and it is possible to achieve sender anonymity using MURBs.

## 3. Layer Overview

The protocol is designed in four layers as shown in Figure 1.

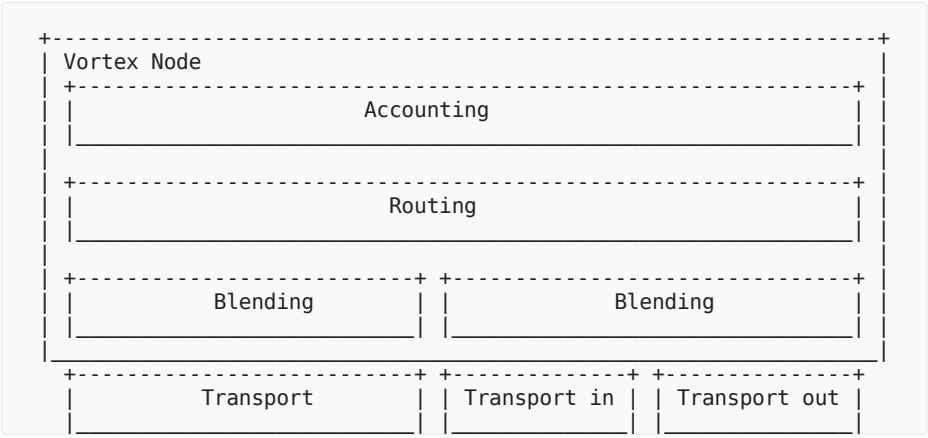


*Figure 1: Layer overview*

Every participating node MUST implement the layer's blending, routing, and accounting. There MUST be at least one incoming and one outgoing transport layer available to a node. All blending layers SHOULD connect to the respective transport layers for sending and receiving packets.

### 3.1. Transport Layer

The transport layer transfers the blended Vortex messages to the next vortex node and stores it until the next blending layer picks up the message.

The transport layer infrastructure SHOULD NOT be specific to anonymous communication and should contain significant portions of non-Vortex traffic.

### 3.2. Blending Layer

The blending layer embeds blended Vortex Message into the transport layer data stream and extracts the packets from the transport layer.

### 3.3. Routing Layer

The routing layer expands the information contained in MessageVortex packets, processes them, and passes generated packets to the respective blending layer.

### 3.4. Accounting Layer

The accounting layer tracks all ephemeral identities authorized to use a MessageVortex node and verifies the available quotas to an ephemeral identity.

## 4. Vortex Message

### 4.1. Overview

Figure 2 shows a Vortex message. The enclosed sections denote encrypted blocks, and the three or four-letter abbreviations denote the key required for decryption. The abbreviation k_h stands for the asymmetric host key, and sk_p is the symmetric peer key. The receiving node obtains this key by decrypting MPREFIX with its host key k_h. Then, sk_s is the symmetric sender key. When decrypting the MPREFIX block, the node obtains this key. The sender key protects the header and routing blocks by

guaranteeing the node assembling the message does not know about upcoming identities, operations, and requests. The peer key protects the message, including its structure, from third-party observers.

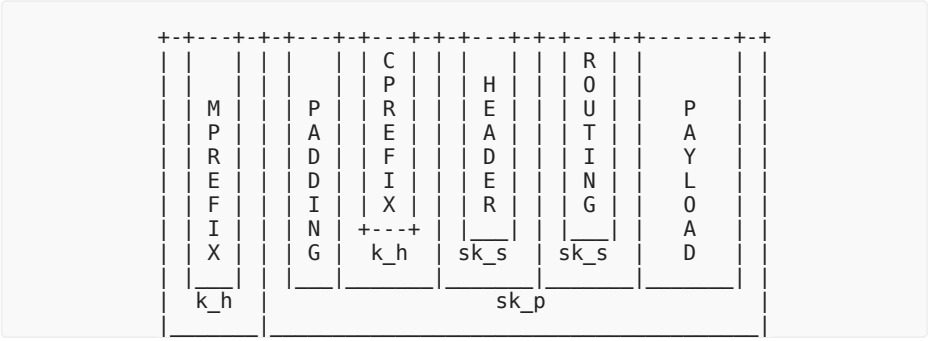


*Figure 2: Vortex message overview*

## 4.2. Message Prefix Block (MPREFIX)

The PrefixBlock contains a symmetrical key as defined in Section a.1 and is encrypted using the host key of the receiving peer host. The symmetric key utilized MUST be from the set advertised by a CapabilitiesReplyBlock (see Section 7.2.6). A node MAY choose any parameters omitted in the CapabilitiesReplyBlock freely unless stated otherwise in Section 7.2.6. A node SHOULD avoid sending unencrypted PrefixBlocks, and a prefix block MUST contain the same forward-secret as the other prefix as well as the routing and header blocks. A host MAY reply to a message with an unencrypted message block, but any reply to a message SHOULD be encrypted.

The sender MUST choose a key which may be encrypted with the host key in the respective PrefixBlock using the padding advertised by the CapabilitiesReplyBlock.

## 4.3. Inner Message Block

A node MUST always encrypt an InnerMessageBlock with the symmetric key of the PrefixBlock to hide the inner structure of the message. The InnerMessageBlock SHOULD always accommodate four or more payload chunks.

An InnerMessageBlock always starts with a padding block, which guarantees that when using the same routing block multiple times, its binary structure is not repeated throughout the messages of the same routing block. The padding MUST be the first 16 bytes of the first four non-empty payload chunks (i.e., PayloadChunks). If a payload chunk is shorter than 16 bytes, then the content of the padding SHOULD be filled with zero-valued bytes (00h) from the end up to the required number of bytes. An inner message block (i.e., InnerMessageBlock) SHOULD contain at least four payload chunks with a size of 16 bytes or larger. If there are less than four payload chunks, then the padding MUST contain a random sequence of 16 bytes for those missing, and a node MUST NOT reuse random sequences.

An InnerMessageBlock contains so-called forwardSecrets, a random number that MUST be the same in the HeaderBlock, RoutingBlock, and PrefixBlock. Nodes receiving messages containing non-matching forwardSecrets MUST discard these messages and SHOULD NOT send an error message. If a node receives too many messages with illegal forward secrets, then the node SHOULD delete this identity. A node receiving a message with a broken forwardSecret SHOULD treat the block as a replayed block and discard it regardless of a valid forwardSecret. Any replay within the replay protection time MUST be discarded regardless of a correct forward secret.

### 4.3.1. Control Prefix Block

Control prefix (CPREFIX) and MPREFIX blocks share the same structure and logic as well as containing the sender key sk_s. If an MPREFIX block is unencrypted, a node MAY omit the CPREFIX block. An omitted CPREFIX block results in unencrypted control blocks (e.g., the HeaderBlock and RoutingBlock).

A prefix block MUST contain the same forwardSecret as the other prefix, the routing block, and the header block.

### 4.3.2. Control Blocks

The control blocks of the HeaderBlock and a RoutingBlock contain the core information to process the payload.

#### 4.3.2.1. Header Block

The header block (see HeaderBlock in Appendix A) contains the following information.

- It MUST contain the local ephemeral identity of the routing block builder.

- It MAY contain header requests.
- It MAY contain the solution to a PuzzleRequired block previously opposed in a header request.

The list of header requests MAY be one of the following.

- Empty.
- Contain a single identity create request (HeaderRequestIdentity).
- Contain a single increase quota request.

If a header block violates these rules, then a node MUST NOT reply to any header request. The payload and routing blocks SHOULD still be added to the workspace and processed if the message quota is not exceeded.

#### 4.3.2.2. Routing Block

The routing block (see RoutingBlock in Appendix A) contains the following information.

- It MUST contain a serial number uniquely identifying the routing block of this user. The serial number MUST be unique during the lifetime of the routing block.
- It MUST contain the same forward secret as the two prefix blocks and the header block.
- It MAY contain assembly and processing instructions for subsequent messages.
- It MAY contain a reply block for messages assigned to the owner of the identity.

### 4.3.3. Payload Block

Each InnerMessageBlock with routing information SHOULD contain at least four PayloadChunks.

# 5. General notes

The MessageVortex protocol is a modular protocol that allows the use of different encryption algorithms. For its operation, a Vortex node SHOULD always support at least two distinct types of algorithms, paddings or modes such that they rely on two mathematical problems.

## 5.1. Supported Symmetric Ciphers

A node MUST support the following symmetric ciphers.

- AES128 (see [FIPS-AES] for AES implementation details).
- AES256.
- CAMELLIA128 (see [RFC3657] Chapter 3 for Camellia implementation details).
- CAMELLIA256.

A node SHOULD support any standardized key larger than the smallest key size.

A node MAY support Twofish ciphers (see [TWOFISH]).

## 5.2. Supported Asymmetric Ciphers

A node MUST support the following asymmetric ciphers.

- RSA with key sizes greater or equal to 2048 ([RFC8017]).
- ECC with named curves secp384r1, sect409k1 or secp521r1 (see [SEC1]).

## 5.3. Supported MACs

A node MUST support the following Message Authentication Codes (MAC).

- SHA3-256 (see [ISO-10118-3] for SHA implementation details).
- RipeMD160 (see [ISO-10118-3] for RIPEMD implementation details).

A node SHOULD support the following MACs.

- SHA3-512.
- RipeMD256.
- RipeMD512.

## 5.4. Supported Paddings

A node MUST support the following paddings specified in [RFC8017].

- PKCS1 (see [RFC8017]).
- PKCS7 (see [RFC5958]).

### 5.5. Supported Modes

A node MUST support the following modes.

- CBC (see [RFC1423]) such that the utilized IV must be of equal length as the key.
- EAX (see [EAX]).
- GCM (see [RFC5288]).
- NONE (only used in special cases, see Section 11).

A node SHOULD NOT use the following modes.

- NONE (except as stated when using the addRedundancy function).
- ECB.

A node SHOULD support the following modes.

- CTR ([RFC3686]).
- CCM ([RFC3610]).
- OCB ([RFC7253]).
- OFB ([MODES]).

## 6. Blending

Each node supports a fixed set of blending capabilities, which may be different for incoming and outgoing messages.

The following sections describe the blending mechanism. There are currently two blending layers specified with one for the Simple Mail Transfer Protocol (SMTP, see [RFC5321]) and the second for the Extensible Messaging and Presence Protocol (XMPP, see [RFC6120]). All nodes MUST at least support "encoding=plain:0,256".

### 6.1. Blending in Attachments

There are two types of blending supported when using attachments.

- Plain binary encoding with offset (PLAIN).
- Embedding with F5 in an image (F5).

A node MUST support PLAIN blending for reasons of interoperability whereas a node MAY support blending using F5.

### 6.1.1. PLAIN embedding into attachments

A blending layer embeds a VortexMessage in a carrier file with an offset for PLAIN blending. For replacing a file start, a node MUST use the offset 0. The routing node MUST choose the payload file for the message, and SHOULD use a credible payload type (e.g., MIME type) with high entropy. Furthermore, it SHOULD prefix a valid header structure to avoid easy detection of the Vortex message. Finally, a routing node SHOULD use a valid footer, if any, to a payload file to improve blending.

The blended Vortex message is embedded in one or more message chunks, each starting with two unsigned integers of variable length. The integer starts with the LSB, and if bit 7 is set, then there is another byte following. There cannot be more than four bytes where the last, fourth byte is always 8 bit. The three preceding bytes have a payload of seven bits each, which results in a maximum number of 2^29 bits. The first of the extracted numbers reflect the number of bytes in the chunk after the length descriptors. The second contains the number of bytes to be skipped to reach the next chunk. There exists no "last chunk" indicator.

```
position:00h   02h   04h   06h   08h ... 400h  402h  404h  406h
408h  40Ah
value:   01 02 03 04 05 06 07 08 09  ... 01 05 0A 0B 0C 0D 0E 0F f0
03 12 13

Embedding: "(plain:1024)"

Result:  0A 13 (+ 494 omitted bytes; then skip 12 bytes to next chunk)
```

A node SHOULD offer at least one PLAIN blending method and MAY offer multiple offsets for incoming Vortex messages.

A plain blending is specified as the following.

```
plainEncoding = "("plain:" <numberOfBytesOfOffset>
                [ "," <numberOfBytesOfOffset> ]* ")"
```

### 6.1.2. F5 embedding into attachments

For F5, a blending layer embeds a Vortex message into a jpeg file according to [F5]. The password for blending may be public, and a routing node MAY advertise multiple passwords. The use of F5 adds approximately tenfold transfer volume to the message. A routing block building node SHOULD only use F5 blending where appropriate.

A blending in F5 is specified as the following.

```
f5Encoding = "(F5:" <passwordString> [ "," <PasswordString> ]* ")"
```

Commas and backslashes in passwords MUST be escaped with a backslash whereas closing brackets are treated as normal password characters unless they are the final character of the encoding specification string.

## 6.2. Blending into an SMTP layer

Email messages with content MUST be encoded with Multipurpose Internet Mail Extensions (MIME) as specified in [RFC2045]. All nodes MUST support BASE64 encoding and MUST test all sections of a MIME message for the presence of a VortexMessage.

A vortex message is present if a block containing the peer key at the known offset of any MIME part decodes correctly.

A node SHOULD support SMTP blending for sending and receiving. For sending SMTP, the specification in [RFC5321] must be used. TLS layers MUST always be applied when obtaining messages using POP3 (as specified in [RFC1939] and [RFC2595]) or IMAP (as specified in [RFC3501]). Any SMTP connection MUST employ a TLS encryption when passing credentials.

## 6.3. Blending into an XMPP layer

For interoperability, an implementation SHOULD provide XMPP blending.

Blending into XMPP traffic is performed using the [XEP-0231] extension of the XMPP protocol.

PLAIN and F5 blending are acceptable for this transport layer.

# 7. Routing

## 7.1. Vortex Message Processing

### 7.1.1. Processing of incoming Vortex Messages

An incoming message is considered initially unauthenticated. A node should consider a VortexMessage as authenticated as soon as the ephemeral identity is known and is not temporary.

For an unauthenticated message, the following rules apply.

- A node MUST ignore all Routing blocks.
- A node MUST ignore all Payload blocks.
- A node SHOULD accept identity creation requests in unauthenticated messages.
- A node MUST ignore all other header requests except identity creation requests.
- A node MUST ignore all identity creation requests belonging to an existing identity.

A message is considered authenticated as soon as the identity used in the header block is known and not temporary. A node MUST NOT treat a message as authenticated if the specified maximum number of replays is reached. For authenticated messages, the following rules apply.

- A node MUST ignore identity creation requests.
- A node MUST replace the current reply block with the reply block provided in the routing block (if any). The node MUST keep the reply block if none is provided.
- A node SHOULD process all header requests.
- A node SHOULD add all routing blocks to the workspace.
- A node SHOULD add all payload blocks to the workspace.

A routing node MUST decrement the message quota by one if a received message is authenticated, valid, and contains at least one payload block. If a message is identified as duplicate according to the reply protection, then a node MUST NOT decrement the message quota.

The message processing works according pseudo-code shown below.

```
function incomming_message(VortexMessage blendedMessage) {
  try{
    msg = unblend( blendedMessage );
    if( not msg ) {
      // Abort processing
      throw exception( "no embedded message found" )
    } else {
      hdr = get_header( msg )
      if( not known_identity( hdr.identity ) {
        if( get_requests( hdr ) contains HeaderRequestIdentity ) {
          create_new_identity( hdr ).set_temporary( true )
          send_message( create_requirement( hdr )  )
        } else {
          // Abort processing
          throw exception( "identity unknown" )
        }
      } else {
        if( is_duplicate_or_replayed( msg ) ) {
          // Abort processing
          throw exception "duplicate or replayed message" )
        } else {
          if( get_accounting( hdr.identity ).is_temporary() ) {
            if( not verify_requirement( hdr.identity, msg ) ) {
              get_accounting( hdr.identity ).set_temporary( false )
            }
          }
          if( get_accounting( hdr ).is_temporary() ) {
            throw exception( "no processing on temporary identity" )
          }

          // Message authenticated
          get_accounting( hdr.identity
).register_for_replay_protection( msg )
          if( not verify_mtching_forward_secrets( msg ) ) {
            throw exception( "forward secret missmatch" )
          }
          if( contains_payload( msg ) ) {
            if( get_accounting( hdr.identity ).decrement_message_quota
() ) {
              while index,nextPayloadBlock = get_next_payload_block
( msg ) {
                add_workspace( header.identity, index,
nextPayloadBlock )
              }
              while nextRoutingBlock = get_next_routing_block( msg ) {
                add_workspace( hdr.identity, add_routing
( nextRoutingBlock ) )
              }
              process_reserved_mapping_space( msg )
              while nextRequirement = get_next_requirement( hdr ) {
                add_workspace( hdr.identity, nextRequirement )
              }
            } else {
              throw exception( "Message quota exceeded" )
```

```
            }
          }
        }
      }
  } catch( exception e ) {
    // Message processing failed
    throw e;
  }
}
```

### 7.1.2. Processing of Routing Blocks in the Workspace

A routing workspace consists of the following items.

- The identity linked to, which determines the lifetime of the workspace.
- The linked routing combos (RoutingCombo).
- A payload chunk space with the following multiple subspaces available:
  - ID 0 represents a message to be embedded (when reading) or a message to be extracted to the user (when written).
  - ID 1 to ID maxPayloadBlocks represent the payload chunk slots in the target message.
  - All blocks between ID maxPayloadBlocks + 1 to ID 32767 belong to a temporary routing block-specific space.
  - All blocks between ID 32768 to ID 65535 belong to a shared space available to all operations of the identity.

The accounting layer typically triggers processing and represents either a cleanup action or a routing event. A cleanup event deletes the following information from all workspaces.

- All processed routing combos.
- All routing combos with expired usagePeriod.
- All payload chunks exceeding the maxProcess time.
- All expired objects.
- All expired puzzles.
- All expired identities.
- All expired replay protections.

Note that maxProcessTime reflects the number of seconds since the arrival of the last octet of the message at the transport layer facility. A node SHOULD NOT take additional processing time (e.g., for anti-UBE or anti-virus) into account.

The accounting layer triggers routing events occurring at least the minProcessTime after the last octet of the message arrived at the routing layer. A node SHOULD choose the latest possible moment at which the peer node receives the last octet of the assembled message before the maxProcessTime is reached. The calculation of this last point in time where a message may be set SHOULD always assume that the target node is working. A sending node SHOULD choose the time within these bounds randomly. An accounting layer MAY trigger multiple routing combos in bulk to further obfuscate the identity of a single transport message.

First, the processing node escapes the payload chunk at ID 0 if needed (e.g., a non-special block is starting with a backslash). Next, it executes all processing instructions of the routing combo in the specified sequence. If an instruction fails, then the block at the target ID of the operation remains unchanged. The routing layer proceeds with the subsequent processing instructions by ignoring the error. For a detailed description of the operations, see Section 7.4. If a node succeeds in building at least one payload chunk, then a VortexMessage is composed and passed to the blending layer.

#### 7.1.3. Processing of Outgoing Vortex Messages

The blending layer MUST compose a transport layer message according to the specification provided in the routing combo. It SHOULD choose any decoy message or steganographic carrier in such a way that the dead parrot syndrome, as specified in [DeadParrot], is avoided.

### 7.2. Header Requests

Header requests are control requests for the anonymization system. Messages with requests or replies only MUST NOT affect any quota.

#### 7.2.1. Request New Ephemeral Identity

Requesting a new ephemeral identity is performed by sending a message containing a header block with the new identity and an identity creation request (HeaderRequestIdentity) to a node. The node MAY send an error block (see Section 7.3.1) if it rejects the request.

If a node accepts an identity creation request, then it MUST send a reply. A node accepting a request without a requirement MUST send back a special block containing "no error". A node accepting a request under the precondition of a requirement to be fulfilled MUST send a special block containing a requirement block.

A node SHOULD NOT reply to any clear-text requests if the node does not want to disclose its identity as a Vortex node officially. A node MUST reply with an error block if a valid identity is used for the request.

#### 7.2.2. Request Message Quota

Any valid ephemeral identity may request an increase of the current message quota to a specific value at any time. The request MUST include a reply block in the header and may contain other parts. If a requested value is lower than the current quota, then the node SHOULD NOT refuse the quota request and SHOULD send a "no error" status.

A node SHOULD reply to a HeaderRequestIncreaseMessageQuota request (see Appendix A) of a valid ephemeral identity. The reply MUST include a requirement, an error message or a "no error" status message.

#### 7.2.3. Request Increase of Message Quota

A node may request to increase the current message quota by sending a HeaderRequestIncreaseMessageQuota request to the routing node. The value specified within the node is the new quota. HeaderRequestIncreaseMessageQuota requests MUST include a reply block, and a node SHOULD NOT use a previously sent MURB to reply.

If the requested quota is higher than the current quota, then the node SHOULD send a "no error" reply. If the requested quota is not accepted, then the node SHOULD send a requestedQuotaOutOfBand reply.

A node accepting the request MUST send a RequirementBlock or a "no error block."

#### 7.2.4. Request Transfer Quota

Any valid ephemeral identity may request to increase the current transfer quota to a specific value at any time. The request MUST include a reply block in the header and may contain other parts. If a requested value is lower than the current quota, then the node SHOULD NOT refuse the quota request and SHOULD send a "no error" status.

A node SHOULD reply to a HeaderRequestIncreaseTransferQuota request (see Appendix A) of a valid ephemeral identity. The reply MUST include a requirement, an error message or a "no error" status message.

#### 7.2.5. Query Quota

Any valid ephemeral identity may request the current message and transfer quota. The request MUST include a reply block in the header and may contain other parts.

A node MUST reply to a HeaderRequestQueryQuota request (see Appendix A), which MUST include the current message quota and the current message transfer quota. The reply to this request MUST NOT include a requirement.

#### 7.2.6. Request Capabilities

Any node MAY request the capabilities of another node, which include all information necessary to create a parseable VortexMessage. Any node SHOULD reply to any encrypted HeaderRequestCapability.

A node SHOULD NOT reply to clear-text requests if the node does not want to disclose its identity as a Vortex node officially. A node MUST reply if a valid identity is used for the request, and it MAY reply to unknown identities.

#### 7.2.7. Request Nodes

A node may ask another node for a list of routing node addresses and keys, which may be used to bootstrap a new node and add routing nodes to increase the anonymization of a node. The receiving node of such a request SHOULD reply with a requirement (e.g., RequirementPuzzleRequired).

A node MAY reply to a HeaderRequest request (see Appendix A) of a valid ephemeral identity, and the reply MUST include a requirement, an error message or a "no error" status message. A node MUST NOT reply to an unknown identity, and SHOULD always reply with the same result set to the same identity.

#### 7.2.8. Request Identity Replace

This request type allows a receiving node to replace an existing identity with the identity provided in the message, and is required if an adversary manages to deny the usage of a node (e.g., by deleting the corresponding transport account). Any sending node may recover from such an attack by sending a valid authenticated message to another identity to provide the new transport and key details.

A node SHOULD reply to such a request from a valid known identity, and the reply MUST include an error message or a "no error" status message.

### 7.3. Special Blocks

Special blocks are payload messages that reflect messages from one node to another and are not visible to the user. A special block starts with the character sequence '\special' (or 5Ch 73h 70h 65h 63h 69h 61h 6Ch) followed by a DER encoded special block (SpecialBlock). Any non-special message decoding to ID 0 in a workspace starting with this character sequence MUST escape all backslashes within the payload chunk with an additional backslash.

#### 7.3.1. Error Block

An error block may be sent as a reply contained in the payload section. The error block is embedded in a special block and sent with any provided reply block. Error messages SHOULD contain the serial number of the offending header block and MAY contain human-readable text providing additional messages about the error.

#### 7.3.2. Requirement Block

If a node is receiving a requirement block, then it MUST assume that the request block is accepted, is not yet processed, and is to be processed if it meets the contained requirement. A node MUST process a request as soon as the requirement is fulfilled, and MUST resend the request as soon as it meets the requirement.

A node MAY reject a request, accept a request without a requirement, accept a request upon payment (RequirementPaymentRequired), or accept a request upon solving a proof of work puzzle (RequirementPuzzleRequired).

#### 7.3.2.1. Puzzle Requirement

If a node requests a puzzle, then it MUST send a RequirementPuzzleRequired block. The puzzle requirement is solved if the node receiving the puzzle is replying with a header block that contains the puzzle block, and the hash of the encoded block begins with the bit sequence mentioned in the puzzle within the period specified in the field 'valid.'

A node solving a puzzle requires sending a VortexMessage to the requesting node, which MUST contain a header block that includes the puzzle block and MUST have a MAC fingerprint starting with the bit sequence as specified in the challenge. The receiving node calculates the MAC from the unencrypted DER encoded HeaderBlock with the algorithm specified by the node. The sending node may achieve the requirement by adding a proofOfWork field to the HeaderBlock containing any content fulfilling the criteria. The sending node SHOULD keep the proofOfWork field as short as possible.

#### 7.3.2.2. Payment Requirement

If a node requests a payment, then it MUST send a RequirementPaymentRequired block. As soon as the requested fee is paid and confirmed, the requesting node MUST send a "no error" status message. The usage period 'valid' describes the period during which the payment may be carried out. A node MUST accept the payment if occurring within the 'valid' period but confirmed later. A node SHOULD return all unsolicited payments to the sending address.

## 7.4. Routing Operations

Routing operations are contained in a routing block and processed upon arrival of a message or when compiling a new message. All operations are reversible, and no operation is available for generating decoy traffic, which may be used through encryption of an unpadded block or the addRedundancy operation.

All payload chunk blocks inherit the validity time from the message routing combos as arrival time + max(maxProcessTime).

When applying an operation to a source block, the resulting target block inherits the expiration of the source block. When multiple expiration times exist, the one furthest in the future is applied to the target block. If the operation fails, then the target expiration remains unchanged.

### 7.4.1. Mapping Operation

The straightforward mapping operation is used in inOperations of a routing block to map the routing block's specific blocks to a permanent workspace.

### 7.4.2. Split and Merge Operations

The split and merge operations allow splitting and recombining message chunks. A node MUST adhere to the following constraints.

- The operation must be applied at an absolute (measuring in bytes) or relative (measured as a float value in the range 0>value>100) position.
- All calculations must be performed according to IEEE 754 [IEEE754] and in 64-bit precision.
- If a relative value is a non-integer result, then a floor operation (i.e., cutting off all non-integer parts) determines the number of bytes.
- If an absolute value is negative, then the size represents the number of bytes counted from the end of the message chunk.
- If an absolute value is greater than the number of bytes in a block, then all bytes are mapped to the respective target block, and the other target block becomes a zero byte-sized block.

An operation MUST fail if relative values are equal to, or less than, zero. An operation MUST fail if a relative value is equal to, or greater than, 100. All floating-point operations must be performed according to [IEEE754] and in 64-bit precision.

### 7.4.3. Encrypt and Decrypt Operations

Encryption and decryption are executed according to the standards mentioned above. An encryption operation encrypts a block symmetrically and places the result in the target block. The parameters MUST contain IV, padding, and cipher modes. An encryption operation without a valid parameter set MUST fail.

### 7.4.4. Add and Remove Redundancy Operations

The addRedundancy and removeRedundancy operations are core to the protocol. They may be used to split messages and distribute message content across multiple routing nodes. The operation is separated into three steps.

1. Pad the input block to a multiple of the key block size in the resulting output blocks.
2. Apply a Vandermonde matrix with the given sizes.
3. Encrypt each resulting block with a separate key.

The following sections describe the order of the operations within an addRedundancy operation. For a removeRedundancy operation, invert the functions and order. If the removeRedundancy has more than the required blocks to recover the information, then it should take only the required number beginning from the smallest. If a seed and PRNG are provided, then the removeRedundancy operation MAY test any combination until recovery is successful.

#### 7.4.4.1. Padding Operation

A processing node calculates the final length of all payload blocks, including redundancy. This is done by L=roof((<input block size in bytes>+4)/<encryption block size in bytes>)*<encryption block size in bytes>. The block is prepended with a 32-bit unit length indicator in bytes (little-endian). This length indicator, i, is calculated by i=<input block size in bytes>*randominteger\cdot L. The remainder of the input block, up to length L, is padded with random data. A routing block builder should specify the value of the $randomInteger$. If not specified the routing node may choose a random positive integer value. A routing block builder SHOULD specify a PRNG and a seed used for this padding. If GF(16) is applied, then all numbers are treated as little-endian representations. Only GF(8) and GF(16) are allowed fields.

For padding removal, the padding i at the start is first removed as a little-endian integer. Second, the length of the output block is calculated by applying <output block size in bytes>=i mod <input block size in bytes>

This padding guarantees that each resulting block matches the block size of the subsequent encryption operation and does not require further padding.

#### 7.4.4.2. Apply Matrix

Next, the input block is organized in a data matrix D of dimensions (inrows, incols) where incols=(<number of data blocks>-<number of redundancy blocks>) and inrows=L/(<number of data blocks>-<number of redundancy blocks>). The input block data is first distributed in this matrix across, and then down.

Next, the data matrix D is multiplied by a Vandermonde matrix V with its number of rows equal to the incols calculated and columns equal to the <number of data blocks>. The content of the matrix is formed by v(i,j)=pow(i,j), where i reflects the row number starting at 0, and j reflects the column number starting at 0. The calculations described must be carried out in the GF noted in the respective operation to be successful. The completed operation results in matrix A.

#### 7.4.4.3. Encrypt Target Block

Each row vector of A is a new data block encrypted with the corresponding encryption key noted in the keys of the addRedundancyOperation. If there are not enough keys available, then the keys used for encryption are reused from the beginning after the final key is used. A routing block builder SHOULD provide enough keys so that all target blocks may be encrypted with a unique key. All encryptions SHOULD NOT use padding.

### 7.5. Processing of Vortex Messages

The accounting layer triggers processing according to the information contained in a routing block in the workspace. All operations MUST be executed in the sequence provided in the routing block, and any failing operation must leave the result block unmodified.

All workspace blocks resulting in IDs of 1 to maxPayloadBlock are then added to the message and passed to the blending layer with appropriate instructions.

## 8. Accounting

### 8.1. Accounting Operations

The accounting layer has two types of operations.

- Time-based (e.g., cleanup jobs and initiation of routing).

- Routing triggered (e.g., updating quotas, authorizing operations, and pickup of incoming messages).

Implementations MUST provide sufficient locking mechanisms to guarantee the integrity of accounting information and the workspace at any time.

#### 8.1.1. Time-Based Garbage Collection

The accounting layer SHOULD keep a list of expiration times. As soon as an entry (e.g., payload block or identity) expires, the respective structure should be removed from the workspace. An implementation MAY choose to remove expired items periodically or when encountering them during normal operation.

#### 8.1.2. Time-Based Routing Initiation

The accounting layer MAY keep a list of when a routing block is activated. For improved privacy, the accounting layer should use a slotted model where, whenever possible, multiple routing blocks are handled in the same period, and the requests to the blending layers are mixed between the transactions.

#### 8.1.3. Routing Based Quota Updates

A node MUST update quotas on the respective operations. For example, a node MUST decrease the message quota before processing routing blocks in the workspace and after the processing of header requests.

#### 8.1.4. Routing Based Authorization

The transfer quota MUST be checked and decreased by the number of data bytes in the payload chunks after an outgoing message is processed and fully assembled. The message quota MUST be decreased by one on each routing block triggering the assembly of an outgoing message.

#### 8.1.5. Ephemeral Identity Creation

Any packet may request the creation of an ephemeral identity. A node SHOULD NOT accept such a request without a costly requirement since the request includes a lifetime of the ephemeral identity. The costs for creating the ephemeral identity SHOULD increase if a longer lifetime is requested.

## 9. Acknowledgments

Thanks go to my family who supported me with patience and countless hours as well as to Mark Zeman for his feedback challenging my thoughts and peace.

## 10. IANA Considerations

This memo includes no request to IANA.

Additional encryption algorithms, paddings, modes, blending layers or puzzles MUST be added by writing an extension to this or a subsequent RFC. For testing purposes, IDs above 1,000,000 should be used.

## 11. Security Considerations

The MessageVortex protocol should be understood as a toolset instead of a fixed product. Depending on the usage of the toolset, anonymity and security are affected. For a detailed analysis, see [MVAnalysis].

The primary goals for security within this protocol rely on the following focus areas.

- Confidentiality
- Integrity
- Availability
- Anonymity
  - Third-party anonymity
  - Sender anonymity
  - Receiver anonymity

These aspects are affected by the usage of the protocol, and the following sections provide additional information on how they impact the primary goals.

The Vortex protocol does not rely on any encryption of the transport layer since Vortex messages are already encrypted. Also, confidentiality is not affected by the protection mechanisms of the transport layer.

If a transport layer supports encryption, then a Vortex node SHOULD use it to improve the privacy of the message.

Anonymity is affected by the inner workings of the blending layer in many ways. A Vortex message cannot be read by anyone except the peer nodes and routing block builder. The presence of a Vortex node message may be detected through the typical high entropy of an encrypted file, broken structures of a carrier file, a meaningless content of a carrier file or the contextless communication of the transport layer with its peer partner. A blending layer SHOULD minimize the possibility of simply detection by minimizing these effects.

A blending layer SHOULD use carrier files with high compression or encryption. Carrier files SHOULD NOT have inner structures such that the payload is comparable to valid content. To achieve undetectability by a human reviewer, a routing block builder should use F5 instead of PLAIN blending. This approach, however, increases the protocol overhead by approximately tenfold.

The two layers of 'routing' and 'accounting' have the deepest insight into a Vortex message's inner working. Each knows the immediate peer sender and the peer recipients of all payload chunks. As decoy traffic is generated by combining chunks and applying redundancy calculations, a node can never know if a malfunction (e.g., during a recovery calculation) was intended. Therefore, a node is unable to distinguish a failed transaction from a terminated transaction as well as content from decoy traffic.

A routing block builder SHOULD follow the following rules not to compromise a Vortex message's anonymity.

- All operations applied SHOULD be credibly involved in a message transfer.
- A sufficient subset of the result of an addRedundancy operation should always be sent to peers to allow recovery of the data built.
- The anonymity set of a message should be sufficiently large to avoid legal prosecution of all jurisdictional entities involved, even if a certain amount of the anonymity set cooperates with an adversary.
- Encryption and decryption SHOULD follow normal usage whenever possible by avoiding the encryption of a block on a node with one key and decrypting it with a different key on the same or adjacent node.
- Traffic peaks SHOULD be uniformly distributed within the entire anonymity set.

- A routing block SHOULD be used for a limited number of messages. If used as a message block for the node, then it should be used only once. A block builder SHOULD use the HeaderRequestReplaceIdentity block to update the reply to routing blocks regularly. Implementers should always remember that the same routing block is identifiable by its structure.

An active adversary cannot use blocks from other routing block builders. While the adversary may falsify the result by injecting an incorrect message chunk or not sending a message, such message disruptions may be detected by intentionally routing information to the routing block builder (RBB) node. If the Vortex message does not carry the information expected, then the node may safely assume that one of the involved nodes is misbehaving. A block building node MAY calculate reputation for involved nodes over time and MAY build redundancy paths into a routing block to withstand such malicious nodes.

Receiver anonymity is at risk if the handling of the message header and content is not done with care. An attacker might send a bugged message (e.g., with a DKIM or DMARC header) to deanonymize a recipient. Careful attention is required when handling anything other than local references when processing, verifying, or rendering a message.

# 12. References

## 12.1. Normative References

**[CCITT.X208.1988]** International Telephone and Telegraph Consultative Committee, "Specification of Abstract Syntax Notation One (ASN.1)", CCITT Recommendation X.208, November 1998.

**[CCITT.X680.2002]** International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", November 2002.

**[EAX]** Bellare, M., Rogaway, P., and D. Wagner, "The EAX mode of operation", 2011.

**[F5]** Westfeld, A., "F5 - A Steganographic Algorithm - High Capacity Despite Better Steganalysis", 24 October 2001.

**[FIPS-AES]** Federal Information Processing Standard (FIPS), "Specification for the ADVANCED ENCRYPTION STANDARD (AES)", November 2011.

**[IEEE754]** IEEE, "754-2008 - IEEE Standard for Floating-Point Arithmetic", 29 August 2008.

**[ISO-10118-3]** International Organization for Standardization, "ISO/IEC 10118-3:2004 -- Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions", March 2004.

**[MODES]** National Institute for Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", December 2001.

**[RFC1423]** Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, DOI 10.17487/RFC1423, February 1993 , <https://www.rfc-editor.org/info/rfc1423>.

**[RFC2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997 , <https://www.rfc-editor.org/info/rfc2119>.

**[RFC3610]** Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003 , <https://www.rfc-editor.org/info/rfc3610>.

**[RFC3657]** Moriai, S. and A. Kato, "Use of the Camellia Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3657, DOI 10.17487/RFC3657, January 2004 , <https://www.rfc-editor.org/info/rfc3657>.

**[RFC3686]** Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)", RFC 3686, DOI 10.17487/RFC3686, January 2004 , <https://www.rfc-editor.org/info/rfc3686>.

**[RFC5234]** Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008 , <https://www.rfc-editor.org/info/rfc5234>.

**[RFC5288]** Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008 , <https://www.rfc-editor.org/info/rfc5288>.

**[RFC5958]** Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010 , <https://www.rfc-editor.org/info/rfc5958>.

**[RFC7253]** Krovetz, T. and P. Rogaway, "The OCB Authenticated-Encryption Algorithm", RFC 7253, DOI 10.17487/RFC7253, May 2014 , <https://www.rfc-editor.org/info/rfc7253>.

**[RFC8017]** Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016 , <https://www.rfc-editor.org/info/rfc8017>.

**[SEC1]** Certicom Research, "SEC 1: Elliptic Curve Cryptography", 21 May 2009.

**[TWOFISH]** Schneier, B., "The Twofish Encryptions Algorithm: A 128-Bit Block Cipher, 1st Edition", March 1999.

**[XEP-0231]** Peter, S.A. and P. Simerda, "XEP-0231: Bits of Binary", 3 September 2008 , <https://xmpp.org/extensions/xep-0231.html>.

## 12.2. Informative References

**[DeadParrot]** Houmansadr, A., Burbaker, C., and V. Shmatikov, "The Parrot is Dead: Observing Unobservable Network Communications", 2013 , <https://people.cs.umass.edu/~amir/papers/parrot.pdf>.

**[KAnon]** Ahn, L., Bortz, A., and N.J. Hopper, "k-Anonymous Message Transmission", 2003.

**[MVAnalysis]** Gwerder, M., "MessageVortex", 2018 , <https://messagevortex.net/devel/messageVortex.pdf>.

**[RFC1939]** Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, DOI 10.17487/RFC1939, May 1996 , <https://www.rfc-editor.org/info/rfc1939>.

**[RFC2045]** Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996 , <https://www.rfc-editor.org/info/rfc2045>.

**[RFC2595]** Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999 , <https://www.rfc-editor.org/info/rfc2595>.

**[RFC3501]** Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003 , <https://www.rfc-editor.org/info/rfc3501>.

**[RFC5321]** Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008 , <https://www.rfc-editor.org/info/rfc5321>.

**[RFC6120]** Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011 , <https://www.rfc-editor.org/info/rfc6120>.

# Appendix A. The ASN.1 schema for Vortex messages

The following sections contain the ASN.1 modules specifying the MessageVortex Protocol.

## A.1. The main VortexMessageBlocks

```
MessageVortex-Schema DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS PrefixBlock, InnerMessageBlock, RoutingBlock,
          maxID;
  IMPORTS SymmetricKey, AsymmetricKey, MacAlgorithmSpec, CipherSpec
                FROM MessageVortex-Ciphers
          HeaderRequest
                FROM MessageVortex-Requests
          PayloadOperation
                FROM MessageVortex-Operations

          UsagePeriod, BlendingSpec
                FROM MessageVortex-Helpers;

  --****************************************************************
  -- Constant definitions
  --****************************************************************

  -- maximum serial number
  maxSerial                 INTEGER ::= 4294967295
  -- maximum number of administrative requests
  maxNumberOfRequests       INTEGER ::= 8
  -- maximum number of seconds which the message might be delayed
  -- in the local queue (starting from startOffset)
  maxDurationOfProcessing   INTEGER ::= 86400
  -- maximum id of an operation
  maxID                     INTEGER ::= 32767
  -- maximum number of routing blocks in a message
  maxRoutingBlocks          INTEGER ::= 127
  -- maximum number a block may be replayed
  maxNumberOfReplays        INTEGER ::= 127
  -- maximum number of payload chunks in a message
  maxPayloadBlocks          INTEGER ::= 127
  -- maximum number of seconds a proof of non revocation may be old
  maxTimeCachedProof        INTEGER ::= 86400
  -- The maximum ID of the workspace
  maxWorkspaceId            INTEGER ::= 65535
  -- The maximum number of assembly instructions per combo
  maxAssemblyInstructions   INTEGER ::= 255

  --****************************************************************
  -- Block Definitions
  --****************************************************************
  PrefixBlock ::= SEQUENCE {
    forwardsecret   ChainSecret,
    key             SymmetricKey,
        version                 INTEGER OPTIONAL
  }

  IdentityBlock ::= SEQUENCE {
    -- Public key of the identity representing this transmission
    identityKey         AsymmetricKey,
    -- serial identifying this block
    serial              INTEGER (0..maxSerial),
```

```
    -- number of times this block may be replayed (Tuple is
    -- identityKey, serial while UsagePeriod of block)
    maxReplays          INTEGER (0..maxNumberOfReplays),
    -- subsequent Blocks are not processed before valid time.
    -- Host may reject too long retention. Recomended validity
    -- support >=1Mt.
    valid               UsagePeriod,
    -- represents the chained secret which has to be found in
    -- subsequent blocks
    -- prevents reassembly attack
    forwardSecret       ChainSecret,
    --  contains the MAC-Algorithm used for signing
    signAlgorithm       MacAlgorithmSpec,
    -- contains administrative requests such as quota requests
    requests            SEQUENCE (SIZE (0..maxNumberOfRequests))
                        OF HeaderRequest ,
    -- Reply Block for the requests
    requestReplyBlock   RoutingCombo,
    -- padding and identitifier required to solve the cryptopuzzle
    identifier  [12201] PuzzleIdentifier OPTIONAL,
    -- This is for solving crypto puzzles
    proofOfWork [12202] OCTET STRING OPTIONAL
  }

  InnerMessageBlock ::= SEQUENCE {
    padding   OCTET STRING,
    prefix    CHOICE {
        plain           [11011] PrefixBlock,
        -- contains prefix encrypted with receivers public key
        encrypted       [11012] OCTET STRING
    },
    identity  CHOICE  {
      -- debug/internal use only
      plain     [11021] IdentityBlock,
      -- contains encrypted identity block
      encyrpted [11022] OCTET STRING
    },
    -- contains signature of Identity [as stored in
    -- HeaderBlock; signed unencrypted HeaderBlock without Tag]
    identitySignature OCTET STRING,
    -- contains routing information (next hop) for the payloads
    routing     CHOICE {
      plain     [11031] RoutingBlock,
      -- contains encrypted routing block
      encyrpted [11032] OCTET STRING
    },
    -- contains the actual payload
    payload     SEQUENCE (SIZE (0..maxPayloadBlocks))
                OF OCTET STRING
  }

  RoutingBlock ::= SEQUENCE {
    -- contains the routingCombos
    routing    [332] SEQUENCE (SIZE (0..maxRoutingBlocks))
                              OF RoutingCombo,
```

```
    -- contains the secret of the header block
    forwardSecret     ChainSecret,
    -- contains a routing block which may be used when sending
    -- error messages back to the quota owner
    -- this routing block may be cached for future use
    replyBlock [131]   SEQUENCE {
         murb           RoutingCombo,
         maxReplay      INTEGER,
         validity       UsagePeriod
    } OPTIONAL
  }

  RoutingCombo ::= SEQUENCE {
    -- contains the period when the payload should be processed
    -- Router might refuse to long queue retention
    -- Recommended support for retention >=1h
    minProcessTime INTEGER (0..maxDurationOfProcessing),
    maxProcessTime INTEGER (0..maxDurationOfProcessing),
    -- The message key to encrypt the message
    peerKey        [401] SymmetricKey OPTIONAL,
    -- contains the next recipient
    recipient      [402] BlendingSpec OPTIONAL,
    -- PrefixBlock encrypted with message key
    mPrefix        [403] OCTET STRING OPTIONAL,
    -- PrefixBlock encrypted with sender key
    cPrefix        [404] OCTET STRING OPTIONAL,
    -- HeaderBlock encrypted with sender key
    header         [405] OCTET STRING OPTIONAL,
    -- RoutingBlock encrypted with sender key
    routing        [406] OCTET STRING OPTIONAL,
    -- contains information for building messages (when used as MURB
    -- ID 0 denotes original message; ID 1-maxPayloadBlocks denotes
    -- target message; 32768-maxWorkspaceId shared workspace for all
    -- blocks of this identity)
    assembly       [407] SEQUENCE (SIZE (0..maxAssemblyInstructions))
                   OF PayloadOperation,
    validity       [408] UsagePeriod,
    -- optional - to identify the sender of a message when received
    id             [409] INTEGER OPTIONAL
  }

  PuzzleIdentifier          ::= OCTET STRING ( SIZE(0..16) )

  ChainSecret ::= INTEGER (0..4294967295)

END
```

## A.2. The VortexMessage Ciphers Structures

```
MessageVortex-Ciphers DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS SymmetricKey, AsymmetricKey, MacAlgorithmSpec,
          MacAlgorithm, CipherSpec, PRNGType;

  CipherSpec ::= SEQUENCE {
      asymmetric [16001] AsymmetricAlgorithmSpec OPTIONAL,
      symmetric  [16002] SymmetricAlgorithmSpec OPTIONAL,
      mac        [16003] MacAlgorithmSpec OPTIONAL,
      cipherUsage[16004] CipherUsage
  }

  CipherUsage ::= ENUMERATED {
    sign     (200),
    encrypt  (210)
  }

  SymmetricAlgorithmSpec  ::=  SEQUENCE  {
    algorithm               [16101]SymmetricAlgorithm,
    -- if ommited: pkcs1
    padding                 [16102]CipherPadding OPTIONAL,
    -- if ommited: cbc
    mode                    [16103]CipherMode OPTIONAL,
    parameter               [16104]AlgorithmParameters OPTIONAL
  }

  AsymmetricAlgorithmSpec  ::=  SEQUENCE  {
    algorithm               AsymmetricAlgorithm,
    parameter               AlgorithmParameters OPTIONAL
  }

  MacAlgorithmSpec  ::=  SEQUENCE  {
    algorithm               MacAlgorithm,
    parameter               AlgorithmParameters
  }

  PRNGAlgorithmSpec ::= SEQUENCE {
    type                    PRNGType,
    seed                    OCTET STRING
  }

  PRNGType ::= ENUMERATED {
    mrg32k3a        (1000),
    blumMicali      (1001)
  }

  SymmetricAlgorithm ::= ENUMERATED {
    aes128          (1000),  -- required
    aes192          (1001),  -- optional support
    aes256          (1002),  -- required
    camellia128     (1100),  -- required
    camellia192     (1101),  -- optional support
    camellia256     (1102),  -- required
    twofish128      (1200),  -- optional support
```

```
    twofish192      (1201),  -- optional support
    twofish256      (1202)   -- optional support
  }

  CipherMode ::= ENUMERATED {
    -- ECB is a really bad choice. Do not use unless really
    -- necessary
    ecb             (10000),
    cbc             (10001),
    eax             (10002),
    ctr             (10003),
    ccm             (10004),
    gcm             (10005),
    ocb             (10006),
    ofb             (10007),
    none            (10100)
  }

  CipherPadding ::= ENUMERATED {
    none            (1000),
    pkcs1           (1001),
    pkcs7           (1002)
  }

  AsymmetricAlgorithm ::= ENUMERATED {
    rsa             (2000),
    dsa             (2100),
    ec              (2200),
    ntru            (2300)
  }

  MacAlgorithm ::= ENUMERATED {
    sha3-256        (3000),
    sha3-384        (3001),
    sha3-512        (3002),
    ripemd160       (3100),
    ripemd256       (3101),
    ripemd320       (3102)
  }

  ECCurveType ::= ENUMERATED{
    secp384r1       (2500),
    sect409k1       (2501),
    secp521r1       (2502)
  }

  AlgorithmParameters ::= SEQUENCE {
    keySize             [10000] INTEGER (0..65535) OPTIONAL,
    curveType           [10001] ECCurveType   OPTIONAL,
    initialisationVector [10002] OCTET STRING  OPTIONAL,
    nonce               [10003] OCTET STRING  OPTIONAL,
    mode                [10004] CipherMode    OPTIONAL,
    padding             [10005] CipherPadding OPTIONAL,
    n                   [10010] INTEGER       OPTIONAL,
    p                   [10011] INTEGER       OPTIONAL,
```

```
    q                    [10012] INTEGER       OPTIONAL,
    k                    [10013] INTEGER       OPTIONAL,
    t                    [10014] INTEGER       OPTIONAL
  }

  -- Symmetric key
  SymmetricKey ::= SEQUENCE {
    keyType SymmetricAlgorithm,
    parameter AlgorithmParameters,
    key     OCTET STRING (SIZE(16..512))
  }

  -- Asymmetric Key
  AsymmetricKey ::= SEQUENCE {
    keyType       AsymmetricAlgorithm,
        -- private key encoded as PKCS#8/PrivateKeyInfo
    publicKey     [2] OCTET STRING,
        -- private key encoded as X.509/SubjectPublicKeyInfo
    privateKey    [3] OCTET STRING OPTIONAL
  }

END
```

## A.3. The VortexMessage Request Structures

```
MessageVortex-Requests DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS HeaderRequest;
  IMPORTS RequirementBlock
                FROM MessageVortex-Requirements
          UsagePeriod, NodeSpec
                FROM MessageVortex-Helpers;

  HeaderRequest ::= CHOICE {
    identity      [0] HeaderRequestIdentity,
    capabilities  [1] HeaderRequestCapability,
    messageQuota  [2] HeaderRequestIncreaseMessageQuota,
    transferQuota [3] HeaderRequestIncreaseTransferQuota,
    quotaQuery    [4] HeaderRequestQuota,
    nodeQuery     [5] HeaderRequestNodes,
    replace       [6] HeaderRequestReplaceIdentity
  }

  HeaderRequestIdentity ::= SEQUENCE {
    period UsagePeriod
  }

  HeaderRequestReplaceIdentity ::= SEQUENCE {
        old                       NodeSpec,
        new            NodeSpec
  }

  HeaderRequestQuota ::= SEQUENCE {
  }

  HeaderRequestNodes ::= SEQUENCE {
    numberOfNodes   INTEGER (0..255)
  }

  HeaderRequestIncreaseMessageQuota ::= SEQUENCE {
    messages INTEGER (0..4294967295)
  }

  HeaderRequestIncreaseTransferQuota ::= SEQUENCE {
    size     INTEGER (0..4294967295)
  }

  HeaderRequestCapability ::= SEQUENCE {
    period UsagePeriod
  }


END
```

## A.4. The VortexMessage Replies Structures

```
MessageVortex-Replies DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS SpecialBlock;
  IMPORTS BlendingSpec, NodeSpec
                FROM MessageVortex-Helpers
          RequirementBlock
                FROM MessageVortex-Requirements
          CipherSpec, PRNGType, MacAlgorithm
                FROM MessageVortex-Ciphers;

  SpecialBlock ::= CHOICE {
    capabilities  [1]  ReplyCapability,
    requirement   [2]  SEQUENCE (SIZE (1..127))
                       OF RequirementBlock,
    quota         [4]  ReplyCurrentQuota,
    nodes         [5]  ReplyNodes,
    status        [99] StatusBlock
  }

  StatusBlock ::= SEQUENCE {
    code            StatusCode
  }

  StatusCode ::= ENUMERATED {

    -- System messages
    ok                          (2000),
    quotaStatus                 (2101),
    puzzleRequired              (2201),

    -- protocol usage failures
    transferQuotaExceeded       (3001),
    messageQuotaExceeded        (3002),
    requestedQuotaOutOfBand     (3003),
    identityUnknown             (3101),
    messageChunkMissing         (3201),
    messageLifeExpired          (3202),
    puzzleUnknown               (3301),

    -- capability errors
    macAlgorithmUnknown         (3801),
    symmetricAlgorithmUnknown   (3802),
    asymmetricAlgorithmUnknown  (3803),
    prngAlgorithmUnknown        (3804),
    missingParameters           (3820),
    badParameters               (3821),

    -- Mayor host specific errors
    hostError                   (5001)
  }

  ReplyNodes ::= SEQUENCE {
      node   SEQUENCE (SIZE (1..5))
             OF NodeSpec
```

```
  }

  ReplyCapability ::= SEQUENCE {
    -- supported ciphers
    cipher            SEQUENCE (SIZE (2..256)) OF CipherSpec,
    -- supported mac algorithms
    mac               SEQUENCE (SIZE (2..256)) OF MacAlgorithm,
    -- supported PRNGs
    prng              SEQUENCE (SIZE (2..256)) OF PRNGType,
    -- maximum number of bytes to be transferred (outgoing bytes in
vortex message without blending)
    maxTransferQuota  INTEGER (0..4294967295),
    -- maximum number of messages to process for this identity
    maxMessageQuota   INTEGER (0..4294967295),
    -- maximum simultaneously tracked header serials
    maxHeaderSerials  INTEGER (0..4294967295),
    -- maximum simultaneously valid build operations in workspace
    maxBuildOps       INTEGER (0..4294967295),
    -- maximum header lifespan in seconds
    maxHeaderLive     INTEGER (0..4294967295),

    supportedBlendingIn SEQUENCE OF BlendingSpec
  }

  ReplyCurrentQuota ::= SEQUENCE {
    messages INTEGER (0..4294967295),
    size     INTEGER (0..4294967295)
  }

END
```

## A.5. The VortexMessage Requirements Structures

```
MessageVortex-Requirements DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS RequirementBlock;
  IMPORTS MacAlgorithmSpec
                FROM MessageVortex-Ciphers
          UsagePeriod, UsagePeriod
                FROM MessageVortex-Helpers;

  RequirementBlock ::= CHOICE {
    puzzle  [1] RequirementPuzzleRequired,
    payment [2] RequirementPaymentRequired
  }

  RequirementPuzzleRequired ::= SEQUENCE {
    -- bit sequence at beginning of hash from encrypted identity
    -- block
    challenge     BIT STRING,
    mac           MacAlgorithmSpec,
    valid         UsagePeriod,
    identifier    INTEGER (0..4294967295)
  }

  RequirementPaymentRequired ::= SEQUENCE {
    account       OCTET STRING,
    ammount       REAL,
    currency      Currency
  }

  Currency ::= ENUMERATED {
    btc      (8001),
    eth      (8002),
    zec      (8003)
  }

END
```

## A.6. The VortexMessage Helpers Structures

```
MessageVortex-Helpers DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  EXPORTS UsagePeriod, BlendingSpec, NodeSpec;
  IMPORTS AsymmetricKey, SymmetricKey
                FROM MessageVortex-Ciphers;

  -- the maximum number of parameters that might be embedded
  maxNumberOfParameter      INTEGER ::= 127

  UsagePeriod ::= CHOICE {
    absolute [2] AbsoluteUsagePeriod,
    relative [3] RelativeUsagePeriod
  }

  AbsoluteUsagePeriod ::= SEQUENCE {
    notBefore       [0]     GeneralizedTime OPTIONAL,
    notAfter        [1]     GeneralizedTime OPTIONAL
  }

  RelativeUsagePeriod ::= SEQUENCE {
    notBefore       [0]     INTEGER OPTIONAL,
    notAfter        [1]     INTEGER OPTIONAL
  }

  -- contains a node spec of a routing point
  -- At the moment either smtp:<email> or xmpp:<jabber>
  BlendingSpec ::= SEQUENCE {
    target            [1] NodeSpec,
    blendingType      [2] IA5String,
    parameter         [3] SEQUENCE ( SIZE (0..maxNumberOfParameter) )
                          OF BlendingParameter
  }

  BlendingParameter ::= CHOICE {
    offset          [1] INTEGER,
    symmetricKey    [2] SymmetricKey,
    asymmetricKey   [3] AsymmetricKey,
    passphrase      [4] OCTET STRING
  }

  NodeSpec ::= SEQUENCE {
    transportProtocol [1] Protocol,
    recipientAddress  [2] IA5String,
    recipientKey      [3] AsymmetricKey OPTIONAL
  }

  Protocol ::= ENUMERATED {
    smtp  (100),
    xmmp  (110)
  }

END
```

## A.7. The VortexMessage Additional Structures

```
-- States: Tuple()=Value() [vallidity; allowed operations] {Store}
-- - Tuple(identity)=Value(messageQuota,transferQuota,sequence of
--   Routingblocks for Error Message Routing) [validity; Requested
--   at creation; may be extended upon request] {identityStore}
-- - Tuple(Identity,Serial)=maxReplays ['valid' from Identity
--   Block; from First Identity Block; may only be reduced]
--   {IdentityReplayStore}

MessageVortex-NonProtocolBlocks DEFINITIONS EXPLICIT TAGS ::=
BEGIN
  IMPORTS PrefixBlock, InnerMessageBlock, RoutingBlock, maxID
                FROM MessageVortex-Schema
          UsagePeriod, NodeSpec, BlendingSpec
                FROM MessageVortex-Helpers
          AsymmetricKey
                FROM MessageVortex-Ciphers
          RequirementBlock
                FROM MessageVortex-Requirements;

  -- maximum size of transfer quota in bytes of an identity
  maxTransferQuota          INTEGER ::= 4294967295
  -- maximum size of message quota in messages of an identity
  maxMessageQuota           INTEGER ::= 4294967295

  -- do not use these blocks for protocol encoding (internal only)
  VortexMessage ::= SEQUENCE {
    prefix        CHOICE {
        plain           [10011] PrefixBlock,
        -- contains prefix encrypted with receivers public key
        encrypted       [10012] OCTET STRING
    },
    innerMessage  CHOICE {
        plain     [10021] InnerMessageBlock,
        -- contains inner message encrypted with Symmetric key from
        -- Prefix
        encrypted [10022] OCTET STRING
    }
  }

  MemoryPayloadChunk ::= SEQUENCE {
    id                        INTEGER (0..maxID),
    payload                   [100] OCTET STRING,
    validity                  UsagePeriod
  }

  IdentityStore ::= SEQUENCE {
    identities SEQUENCE (SIZE (0..4294967295))
               OF IdentityStoreBlock
  }

  IdentityStoreBlock ::= SEQUENCE {
    valid               UsagePeriod,
    messageQuota        INTEGER (0..maxMessageQuota),
    transferQuota       INTEGER (0..maxTransferQuota),
```

```
    -- if omitted this is a node identity
    identity            [1001] AsymmetricKey OPTIONAL,
    -- if ommited own identity key
    nodeAddress         [1002] NodeSpec      OPTIONAL,
    -- Contains the identity of the owning node;
    -- May be ommited if local node
    nodeKey             [1003] SEQUENCE OF AsymmetricKey OPTIONAL,
    routingBlocks       [1004] SEQUENCE OF RoutingBlock OPTIONAL,
    replayStore         [1005] IdentityReplayStore,
    requirement         [1006] RequirementBlock OPTIONAL
  }

  IdentityReplayStore ::= SEQUENCE {
    replays   SEQUENCE (SIZE (0..4294967295))
              OF IdentityReplayBlock
  }

  IdentityReplayBlock ::= SEQUENCE {
    identity            AsymmetricKey,
    valid               UsagePeriod,
    replaysRemaining  INTEGER (0..4294967295)
  }

END
```

## Author's Address

University of Applied Sciences of
Northwestern Switzerland
**Martin Gwerder**
Bahnhofstrasse 5
CH-5210 Windisch
Switzerland
Phone: +41 56 202 76 81
Email: rfc@messagevortex.net

# A Bibliography

# A Short Biography

Martin Gwerder was born 20. July 1972 in Glarus, Switzerland. He is currently a doctoral student at the University of Basel. After having concluded his studies at the polytechnic at Brugg in 1997, he did a postgraduate education as a master of business and engineering. Following that, he changed to the university track doing an MSc in Informatics at FernUniversität in Hagen. While doing this, he steadily broadened his horizon by working for industry, banking, and government as an engineer and architect in security-related positions. He currently holds a lecturer position for cloud and security at the University of Applied Sciences Northwestern Switzerland. His primary expertise is in the field of networking-related problems dealing with data protection, distribution, confidentiality, and anonymity.

,