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
December 6, 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

1 Foreword

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 900'000 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.

This work splits into multiple parts. In the first part, we collect available researches and technologies. We emphasize techniques on the strength and weaknesses relevant to this work.

In the second part, we reassemble the pieces to a new protocol.

In the third part, we analyze the protocol for the fitness of the purpose. We try to find weaknesses and work out recommendations for protocol usage.

In the last part, we discuss the results and try to summarize the findings. We furthermore elaborate to what extent the protocol fulfills the requirements mentioned in the previous sections.

1.1 Contributions

This thesis contributes to the topic in the following senses:

- It introduces a consistent model for message delivery, which includes all endpoints and involved parties.
- It shows an approach based on existing protocols for anonymous communication, which gives full control of the anonymity to the sender while controlling the costs.
- It offers a client application implementing the proposed Protocol as IMAPv4 cache daemon and as SMTP relay.

2 Notation

2.1 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.

2.2 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.
```

2.3 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.

3 Main Research Question

The main topic of this thesis was defined as follows:

- Is it possible to have a specialized 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? (SQ1)
2. How can entities utilizing *MessageVortex* be attacked, and what measures are available to circumvent such attacks? (SQ2)
3. How can design mitigate attacks target anonymity of a sending or receiving entity within *MessageVortex*? (SQ3)

3.1 SQ1: 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 previously.

Main results of this question are found in part II and part III.

3.2 SQ2: 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 elaborate if those attacks might be successful (and if so under what circumstances).

We discuss this question in part IV.

3.3 SQ3: 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 SQ1.

This question is answered in part IV.

Part II

Methodes

In this part of the thesis, we collect requirements, definitions, methods, and existing research relevant to the topic of this thesis. We explain the choices made and what solutions have been discarded.

We start by collecting requirements for the protocol. Having the requirements, we collect existing technologies on research and implementation levels. Each of the techniques is quickly categorized and either further studied or rejected, naming the reasons for rejection.

The list of technologies and research collected is big. All relevant technologies, either widely adopted or thoroughly researched, should be included in this chapter. All Technologies and research are categorized. We reference technologies through their standards. If applicable, multiple standards may be part of the analysis. A short introduction into the protocol is given and then analyzed for suitability for a specific problem addressed in this work. Whenever quoting research, we refer to the respective papers. If appropriate, multiple related pieces of research are collected together into a bigger picture and then analyzed. When analyzing, we focus on suitability concerning specific problems. If related to standard technology, we link to the respective standards. Transport layer protocols have been shortened to the outcomings in table 5.1. For an extensive analysis see appendix B

In chapter 4, we collect the requirements of the protocol. Based on the findings in this section, we collect in chapter 5. In chapter 6, we collect relevant research about the topic. In chapter 7, we summarize the chosen methods. We explain choices and give a rough outline of the protocol working.

4 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 4.1: 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.

4.1 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.

4.2 Required Properties of an unobservable network

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

4.2.1 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 6.5.4.2 and 6.5.1. We have seen attacks on such systems in the past. Some of them were successful.

4.2.2 Censorship Resistant

In our scenario in 4, 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.

4.2.3 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.

4.2.4 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**].

4.2.5 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**]).

4.2.6 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.

4.2.7 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 4.3.1). 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.

4.3 Outline of Protocol

To fulfill the criteria given above, we outline our idea of the protocol and its layers. We give an overview in figure 4.2. The protocol works on the base of onion routing. Unlike Tor (see 6.6.11), it does not rely on central, censorable infrastructures. To hide message sizes, we designed the system in such a way that it is not limited to onionized messages. Instead, it is capable of splitting and reassemble messages at any intermediate router.

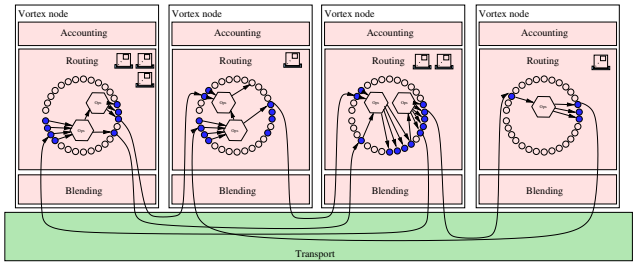

Figure 4.2: A generic overview for the protocol showing a circular path of a message

The protocol itself sends messages through a well-known transport protocol (“Transport”). We use this protocol for transporting our messages to the next peer. The messages are hidden within regular messages already using that transport layer. In an ideal implementation, any messages sent by our protocol should be indistinguishable from regular messages (by computers and humans). To achieve this steganographic task, we introduce a “blending layer”. This layer usually is specific to the underlying transport layer and may vary. We will describe our blending layer implementation in section 7.2.4.

The “routing layer” is the layer that receives and sends messages. It parses only the core protocol, and the data processed here is entirely independent of the underlying transport and blending layer. Our routing layer processes payloads received from VortexMessages with a limited set of predefined operations. The sending node chooses the operations applied to the message.

The “accounting layer” disassembles the received messages into its parts. It then recomposes and stores the messages for further routing. We introduce accounting to avoid that unsolicited bulk messages (UBM) may be transmitted over our media. This accounting additionally prevents a sender from using up too many resources of any intermediate node. It enables any node to keep control of its resource usage and keeps other nodes from eating up more resources than planned.

The messages passed through such a protocol stack are onionised to offer anonymity from evil nodes. Furthermore, any node or external observer is unable to tell whether a message was intended for a specific other node or not. In consequence, we have to guarantee that all nodes, including sender and recipient, offer the same functions and are indistinguishable.

Operations applied always result in a valid message. As an onionised packet typically loses data on its way, the operations must increase and decrease the message in size to mitigate size tracking attacks. The operations must not leak, which part is a decoy or real parts of the message. Operations not carried out due to missing data might be either intentionally or a malfunction.

The defined message operations are as follows:

- Split a message block into two parts of variable size or join them.
  We use this operation to fan out decoy traffic, to cut off decoy data, or to send multiple parts of a message through different routing nodes.
- Encrypt or decrypt a block with a given key.
- Add and remove redundancy information to a stream
  We use this operation to send a message or decoy through multiple channels or recover a partially received message according to the redundancy operation.

To avoid making this system attractive to UBM and to protect DoS attacks, we introduce accounting to make a mass transfer of data more expensive than other means of message transport. The system works as follows:

- Routing nodes may offer their services at a cost. A participating node may pay a fee or solve a crypto-puzzle to pay for these costs.
- The following operations may be cost-effective:
  - Getting an ephemeral identity (eID) on a node (prerequisite for accounting; See section 4.3.1).
  - Assign a maximum number of messages and bytes in a specified interval to an eID on a node.
  - Offer information about the known routing network by the node.
  - Offer information about the current quota of an eID.

As we want to keep resource consumption for accounting as low as possible, all quotas are limited to a time interval. Such time-bound quotas guarantee that quota data of an accounting layer does not grow indefinitely. It is up to the node to decide what time duration is still acceptable.

We minimize the possibility of an overload attack against a node. We achieve this with the following precautions:

- The message build process is far more complex than the routing carried out by the nodes.
- Messages may be processed in parts to minimize the amount of work. After processing the start of a message, a node may already decide whether it will route the message or not.
- The protocol minimizes the use of inefficient operations (e.g., asymmetric encryption).
- The server may limit specific operations.

4.3.1 Ephemeral Identity

We use accounting on various levels. While we are dealing with anonymity, accounting has still to be linked to some identity allowing us to detect malicious actions and ban misbehaving behavior. For this reason, we are introducing a term called “ephemeral identity” (eID).

> An eID is a temporary identity, which is defined by the following attributes:
>
> - It is an identity represented by a public key.
> - It is only valid for a short, limited time-span.
> - It is not linkable to another identity of any kind. It is not linkable, especially to the senders’ known identity unless this sender is trustworthy, and the sender trusts the infrastructure. Such a trust would always be an exception and is not a prerequisite for successful message transfer.

The key to this definition is the last point. It is crucial and, at the same time, hard to achieve. In the protocol, any node may request from another node to accept a new eID with a specified quota and a limited lifespan. The node accepting the identity may link its acceptance to the solving of a proof of work puzzle or paying a micro fee in digital currency.

As the eIDs are limited in their lifespan, a single user has at least one probably multiple eIDs on any node used directly or indirectly for routing.

4.4 Draft of VortexMessages

For our protocol, we assume the following outline for a message:

- Header block
  This block contains an eID of the sender on the message processing host. It allows the host to decide whether he is willing to process the rest of the message or not. It is essential to know that the identity block contains a symmetric key for decryption of the main block and a secret repeated in the main block. This secret, shared among blocks, keeps a malicious node from exchanging any block.
- Routing Blocks
  This block contains routing information for all blocks. It furthermore may contain instruction for processing the data blocks and routing/reply blocks for subsequent processing. Routing blocks are not necessarily related to the payload in the same message. They may pick up payloads from other blocks.
- payload Block
  These Blocks form the payload data. They might contain a message, parts of a message, or decoy material in an unreadable and unidentifiable form.

The message is picked up from the transport layer by the blending layer. This layer extracts the contained data with the help of a host key and passes it to the routing layer. The routing layer extracts the message by decrypting the header block. The accounting layer is called to authorize further processing for the identity. If approved, the routing layer adds the routing blocks to the identity-specific workspace. As soon as the time specified in the routing block arrises, the routing block is processed by following the operations specified within. New messages are assembled and sent. The accounting layer keeps track of the eIDs quota concerning the newly created messages.

5 Transport Layer Protocol analysis

In this chapter, we look at the various transport layer protocols. The primary goal is to identify strong candidates as a transport layer. For a full analysis see B.

5.1 Crtieria

When evaluating the transport protocols, we first compiled a list of common messaging protocols. We then analyze these protocols for the following criteria:

- Widely adopted (Ct1)
  The more widely adopted and used a protocol is, the harder it is for an adversary to monitor (due to the sheer mass), filter, or block the protocol (censorship resistance).
- Reliable (Ct2)
  Message transport between peers should be reliable. As messages may arrive anytime from everywhere, we do not have means to synchronize the peer partners on a higher level without investing a considerable effort. To avoid this effort, we do look for inherently reliable protocols.
- Symmetrical built (Ct3)
  The transport layer should rely on a peer to peer base. All servers implement a generic routing that requires no prior knowledge of all possible targets. This criterion neglects centralized infrastructures. This criterion may be dropped, assuming that the blending layer or a specialized transport overlay is responsible for routing.

5.2 Evaluation Result for Transport Protocols

Protocol \ Criteria	Ct1: Widely adopted	Ct2: Reliable	Ct3: Symmetrically built
HTTP	✓	✓	×
FTP	✓	✓	×
TFTP	×	×	×
MQTT	~	✓	×
AMQP	~	✓	×
CoAP	~	~	×
WAMP	×	✓	~
XMPP	✓	✓	✓
SMTP	✓	✓	✓

Table 5.1: comparison of protocols in terms of the suitability as transport layer

Table 5.1 sums up all previously analyzed protocols in section B. We use “✓” for a fulfilled criterion, “~” for a partially fulfilled criterion, and “×” for a not fulfilled criterion. This overview shows in compact form protocols identified as strong candidates for use as a transport layer in terms of an anonymizing protocol.

This table shows that strong identified candidates are SMTP (being already a message sending protocol on asynchronous base) and XMPP (a real-time chat protocol able to attach files). Both have the advantages that they are widely adopted on the Internet and do additional support content (such as alternatives or attachments).

If assuming an implemented transport layer overlay on the MessageVortex protocol side, HTTP and FTP are suitable candidates as well.

6 Existing Research and Implementations on the Topic

6.1 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.

6.1.1 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.

6.1.2 k-Anonymity

k-anonymity is a term introduced in [**k-anonymous:ccs2003**]. This work claims that entities are not responsible for an action if an observer is unable to match a specific action to less than k entities.

The Document distinguishes between *Sender k-anonymity*, where the sending entity can only be narrowed down to a set of k entities and *Receiver k-anonymity*.

The size of k is a crucial factor. One of the criteria is the legal requirements of the jurisdiction. Depending on the jurisdiction, it usually is not possible to prosecute someone if an action is not directly coupled to one person. Another criterion might be the decreasing of k over time. If a Vortex account is used, we have to assume that some vortex identities go out of commission over time. If k is chosen according to a legal requirement, it should be taken into account that k might be decreasing over time.

6.1.3 ℓ-Diversity

In [**machanavajjhala2007diversity**] an extended model of k-anonymity is introduced. In this paper, the authors emphasize that it is possible to break a k-anonymity set if there is additional information available which may be merged into a data set so that a distinct entity can be filtered from the k-anonymity set. In other words, if an anonymity set is to tightly specified, additional background information might be sufficient to identify a specific entity in an anonymity set.

It might be arguable that a k-anonymity in which a member is not implicitly k-anonymous still is sufficient for k-anonymity in its sense. However, the point made in this work is right and is taken into account. Their approach is to introduce an amount of invisible diversity into k-anonymous sets, so that common background knowledge is no longer sufficient to isolate a single member.

6.1.4 t-Closeness

While ℓ-diversity protects the identity of an entity, it does not prevent information gain. A subject which is in a class has the same attributes. This is where t-closeness[**li2007t**] comes into play. t-closeness is defined as follows:

> An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold. A table is said to have t-closeness if all equivalence classes have t-closeness.

6.2 Single Use Reply Blocks and Multi-Use Reply Blocks

Chaum first introduced the use of reply blocks in [**CHAUM1**]. A routing block, in general, is a structure allowing to send a message to someone without knowing the targets' real address. Reply blocks may be differentiated into two classes "Single Use Reply Blocks" (SURBs) and "Multi-Use Reply Blocks" (MURBs). SURBs may be used once while MURBs may be used a limited number of times.

Within our research, we discovered that if a routing protocol is reproducible, the traffic of a MURB may be used to identify some of the properties of the message. Depending on the type of attack, the block has to be repeated very often. For this reason, we limited the number of replays to a low number. The concept is that we have, in our case a routing block, which might be used up to n times $(0 < n < 127)$. It is easily representable in a byte integer (signed or unsigned) on any system. It is big enough to support human communication sensibly and is big enough to add not too much overhead when rerequesting more MURBs. The number should not be too big because if a MURB is reused, the same pattern of traffic is generated, thus making the system susceptible to statistical attacks.

6.3 Censorship

As a definition for censorship we take

> Censorship: the cyclical suppression, banning, expurgation, or editing by an individual, institution, group or government that enforce or influence its decision against members of the public – of any written or pictorial materials which that individual, institution, group or government deems obscene and "utterly without redeeming social value," as determined by "contemporary community standards."

The definition is attributed to Chuck Stone Professor at the School of Journalism and Mass Communication, University of North Carolina. Please note that "Self Censorship" (not expressing something in fear of consequences) is a form of censorship too.

In our more technical we reduce the definition to

> Censorship: A systematic suppression, modification, or banning of data in a network by either removal, or modification of the data, or systematic influencing of entities involved in the processing (e.g., by creating, routing, storing, or reading) of this data.

This simplified definition narrows down the location to the Internet as it is the only relevant location for us. Furthermore, it limits the definition to the maximum reach within that system.

6.3.1 Censorship Resistant

A censorship-resistant system is a system that allows the entities of the system and the data itself to be unaffected from censorship. Please note that this does not deny the presence of censorship per se. It still exists outside the system. However, it has some consequences for the system itself.

- The system must be either undetectable or out of reach for an entity censoring.
  The possibility of identifying a protocol or data allows a censoring entity to suppress the use of the protocol itself.
- The entities involved in a system must be untraceable.
  Traceable entities would result in a mean of suppressing real-world entities participating in the system.

6.3.2 Parrot Circumvention

In [**oakland2013-parrot**] **oakland2013-parrot** express that it is easy for a human to determine decoy traffic as the content is easily identifiable as generated content. While this is true, there is a possibility here to generate "human-like" data traffic to a certain extent.

[1] footnotes omitted in quote

As an adversary may not assume that his messages are replied to, the problem does not boil down to a true Turing test. It remains on a "passive observer Turing test", enabling the potential nodes to choose their messages.

In our design, this is the job covered by the blending layer. The blending layer generates these messages. These messages are context-less or remain in the context of previous conversations.

6.3.3 Censorship Circumvention

Several technical ways have been explored to circumvent censorship. All seem to boil down to the following main ideas:

- Hide data
- Copy or distribute data to a vast amount of places to improve the lifespan of data
- Outcurve censorship measurements

In the following section, we look at technologies and ideas dealing with these circumvention technologies.

6.3.3.1 Covert Channel and Channel Exploitations

The original term of covert channels was defined by **Lampson73anote**[**Lampson73anote**] as

> not intended for information transfer at all, such as the service program's effect on system load.

This was defined in such a way to distinguish the message flow from

> legitimate channels used by the confined service, such as the bill.

The use of a legitimate channel such as SMTP and hide information within this specific channel is not a usage of a covert channel. We refer to this as channel exploitation.

6.3.3.2 Steganography

Steganography is an important part when it comes to unlinking information. In [**6828087**] and [**subhedar2014current**] we get a very rough overview. As some of the types and algorithms address specific topics of steganography (e.g., some hide from automatic detection and others address a human message stream auditor), we need to choose carefully. In our specific case, the main idea is to hide within the sheer mass of Internet traffic. As a human auditor screening all the messages is a minor thread, we focus on machine-based censorship. Most of the images sent in SMTP are jpg images (see table 9.2 on page 45). We limited our search to algorithms capable of hiding binary data within these files. The number of academically researched options was surprisingly low.

6.3.3.3 Timing Channels

Timing channels are a specialized form of covert channels. In timing channels, the information itself hides not within the data of the channel, but the usage of the channel is in such a way that it is capable of reflecting the data. As we do not have control over the timing of the transport channel, this is not an option for us.

6.4 Cryptography

Whenever dealing with obfuscating data and maintaining the integrity of data, cryptography is the first tool in the hand of an implementer. A vast amount of research in this area does already exist. For this work, we focussed on algorithms either very well researched and implemented or research, which seem very valuable when putting this work into place.

In symmetric encryption in this paper always assumes that

$$D^{K_a}\left(E^{K_a}\left(\mathbf{M}\right)\right) = \mathbf{M} \tag{6.1}$$

For a key $K_b \neq K_a$ this means

$$D^{K_a}\left(E^{K_b}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.2}$$

$$D^{K_b}\left(E^{K_a}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.3}$$

The following candidates have been analyzed:

- AES
  NIST announced AES in **standard2001announcing** as a result of a contest. The algorithm works with four operations (subBytes, ShiftRows, mixColumns, and addRoundKey). These operations are repeated depending on the key length 10 to 14 times.

  AES is up until now (2018) unbroken. It has been weakened in the analysis described in [**tao2015improving**], which reduces the complexity by roughly one to two bits.
- Camellia
  The camellia algorithm is described in [**RFC3713**]. The key sizes are 128, 192, and 256. Camellia is a Feinstel cipher with 18 to 24 rounds depending on the key size. Up until today, no publication claims break this cipher.

For all asymmetric encryption algorithm in this paper, we may assume that. . .

$$D^{K_a^{-1}}\left(E^{K_a^1}\left(\mathbf{M}\right)\right) = \mathbf{M} \tag{6.4}$$

$$D^{K_a^1}\left(E^{K_a^{-1}}\left(\mathbf{M}\right)\right) = \mathbf{M} \tag{6.5}$$

It is important that

$$D^{K_a^{-1}}\left(E^{K_a^{-1}}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.6}$$

$$D^{K_a^1}\left(E^{K_a^1}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.7}$$

And for any other Keypair $K_a^p \neq K_b^p$

$$D^{K_b^{-1}}\left(E^{K_a^1}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.8}$$

$$D^{K_b^1}\left(E^{K_a^1}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.9}$$

$$D^{K_b^{-1}}\left(E^{K_a^{-1}}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.10}$$

$$D^{K_b^1}\left(E^{K_a^{-1}}\left(\mathbf{M}\right)\right) \neq \mathbf{M} \tag{6.11}$$

The number of crypto algorithms was higher than the steganography options. When looking for well-researched algorithms basing on different mathematical problems and having well-defined outlines, numbers dropped dramatically again.

- RSA
  In **Rivest:1978:MOD:359340.359342** the authors **Rivest:1978:MOD:359340.359342** published with [**Rivest:1978:MOD:359340.359342**] a paper which did revolutionize cryptography for years. In their paper, the authors described an encryption method later to be called RSA, which required a key pair (K_a) referenced as public (K_a^1) and private keys (K_a^{-1}). The novelty of this system was that anything encrypted with the public key was only decryptable with the private key and vice versa.

RSA is up until the day of writing this paper not publicly know to be broken (unless a too small key size is used). However – **Shor97polynomial-timealgorithms** described in **Shor97polynomial-timealgorithms** an algorithm which should enable quantum computers to break RSA far faster than done with traditional computers. In the section 6.4.3 we do elaborate these effects further.

- ECC
  The elliptic curves were independently suggested by [**Miller1986**] and [**Koblitz04guideto**] in 1986. Elliptic curve Cryptography started to be widely deployed in the public space in 2006. Since then, it seems to compete very well with the well established RSA algorithm. While being similarly well researched ECC, has the advantage of far shorter key sizes for the same grade of security.

- McElliece
  McEliece was first implemented and then removed again. The key size to gain equivalent security to RSA1024 was $\approx 1MB$. This was impractical and thus discarded again. This was done, although there is up until now no known quantum capable algorithm reducing the key size of McEliece.

- NTRU
  In [**Hoffstein1998**] **Hoffstein1998** described the NTRU algorithm. The inclusion of this algorithm was disputed as it is patented in the united states as US7031468. It was included because the company Security Innovation holding the patent, released the NTRU algorithm on March [2] 2018 into the public domain according to a blog entry on the company website. While NTRU is not as well researched as RSA, it has been around for more than 20 years without being significantly affected by known attacks.

- ElGammal
  We rejected ElGamal as a cryptosystem to include. It bases on the same mathematical problems for cryptoanalysis as RSA (discreet logarithms) but is not as common as RSA.

6.4.1 Homomorphic encryption

Homomorphic encryption, as introduced in [**feldman1987practical**], was from the beginning a strong candidate to be used within our work. Unfortunately, we did not find a way to apply the core addRedundancy operation in homomorphic encryption. Transforming the original data to the GF space in an efficient way to apply matrices was not doable and thus rejected.

6.4.2 Deniable Encryption and Deniable Steganography

Deniable encryption and deniable steganography have been considered out-of-bounds for this work. The main reason is that the presence of encryption (which is not deniable in both cases) may be sufficient for a censor to block a message. Adding a layer to make sure that encryption or steganography is deniable, does not add valuable properties to our system as the sheer presence of encryption might be sufficient for censorship.

6.4.3 Key Sizes

The question of key sizes is hard to answer as it depends on the current and future possibilities of an adversary, which is again depending on not foreseeable research. We tried to collect a couple of recommendations.

Encrypt II (http://www.ecrypt.eu.org/) recommends currently for a "foreseeable future" 256 Bits for symmetric encryption and for asymmetric encryption based on factoring modulus 15424 Bits. Elliptic Curve Cryptography and Hashing should be sufficient if used with at least 512 Bits. If the focus is reduced to the next $\approx$ 20 years, then the key size recommendations are reduced to 128 Bit for symmetric encryption, 3248 Bits for factoring modulus operations, and 256 Bits for elliptic curves and hashing.

According to the equations proposed by **Lenstra04keylength.** in [**Lenstra04keylength.**] an asymmetric key size of 2644 Bits respectively symmetric key length of 95 Bits, or 190 Bits for elliptic curves and hashing should be sufficient for security up to the year 2048.

According to [**CNSASuite**] (superseding well known and often used [**nsa-fact-sheet-B**]) data classified up to "top secret" should be signed with RSA 3072+ or ECDSA P-384. For symmetric encryption, they recommend AES 256 Bits, for Hashing at least SHA-384 and for Elliptic curves a 384 Bit sized key.

As it might seem not a wise idea to consider the recommendation of a potential state-sponsored adversary and the Formulas proposed by **Lenstra04keylength.** do not explicitly take quantum computers into account, we follow the advice of ENCRYPT II.

Furthermore, taking all recommendations together, it seems that all involved parties assume the most trust in elliptic curves rather than asymmetric encryption based on factoring modulus.

6.4.4 Cipher Mode

The cipher mode defines how multiple blocks encrypted with the same key are handled. Main characteristics of cipher modes to us are:

- Parallelisable
  Can multiple parts of a plaintext be encrypted simultaneously? This feature is important for multi CPU and multi-core systems as they can handle parallelizable more efficiently by distributing them on multiple CPUs.

- Random access in decryption
  Random access on decryption allows efficient partial encryption of a ciphertext.

- Initialisation vector
  An initialization vector has downsides and advantages. On the downsides is the fact that an initialization vector must be shared with the message or before distributing it. It is essential to understand that the initialization vector itself usually is not treated as a secret. It is not part of the key.

- Authentication
  Authentication guarantees that the deciphered plaintext has been unmodified since encryption. It does not make a statement over the identity of the party encrypting the text. Such an identifying authentication is referred to as signcryption.

We evaluated the most common cipher modes for suitability. For MessageVortex, we focussed on modes that have the properties parallelizable, random access, and do not do authentication. The main focus, besides the characteristics mentioned above, was on the question of whether there is an open implementation available in java, which is reasonably tested.

- ECB (Electronic Code Book)
  ECB is the most basic mode. Each block of the cleartext is encrypted on its own. This results in a big flaw: blocks containing the same data will always transform to the same ciphertext. This property makes it possible to see some structures of the plain text when looking at the ciphertext. This solution allows the parallelization of encryption, decryption, and random access while decrypting. Due to these flaws, we rejected this mode.

- CBC (Cypher Block Chaining)
  CBC extends the encryption by xor'ing an initialization vector into the first block before encrypting. For all subsequent blocks, the ciphertext result of the preceding block is taken as xor input. This solution does not allow parallelization of encryption, but decryption may be paralleled, and random access is possible. As another downside, CBC requires a shared initialization vector. As with most IV bound modes, an IV/key pair should not be used twice, which has implications for our protocol.

- PCBC (Propagation Cypher Block Chaining)
  CBC extends the encryption by xor'ing, not the ciphertext but a xor result of ciphertext and plaintext. This modification denies parallel decryption and random access compared to CBC.

- EAX
  EAX has been broken in 2012[**minematsu2013attacks**] and is therefore rejected for our use.

- CFB (Cypher Feedback) CFB is specified in [**dworkin2001recommendation**] and works precisely as CBC with the difference that the plain text is xor'ed and the initialization vector, or the preceding cipher result is encrypted. CFB does not support parallel encryption as the ciphertext input from the preceding operation is required for an encryption round. CFB does, however, allow parallel decryption and random access.

- OFB
  [**dworkin2001recommendation**] specifies OFB and works exactly as CFB except for the fact that not the ciphertext result is taken as feedback but the result of the encryption before xor'ing the plain text. This denies parallel encryption and decryption, as well as random access.

- OCB (Offset Codebook Mode)
  This mode was first proposed in [**rogaway2003ocb**] and later specified in [**krovetz-ocb-04**]. OCB is specifically designed for AES128, AES192, and AES256. It supports authentication tag lengths of 128, 96, or 64 bits for each specified encryption algorithm. OCB hashes the plaintext of a message with a specialized function $H_{OCB}(\mathbf{M})$. OCB is fully parallelizable due to its internal structure. All blocks except the first and the last can be encrypted or decrypted in parallel.

- CTR
  CTR is specified in [**lipmaa2000ctr**] and is a mixture between OFB and CBC. A nonce concatenated with a counter incrementing on every block is encrypted and then xor'ed with the plain text. This mode allows parallel decryption and encryption, as well as random access. Reusing IV/Key-pairs using CTR is a problem as we might derive the xor'ed product of two messages. This problem only applies where messages are not uniformly random such as in an already encrypted block.

- CCM
  Counter with CBC-MAC (CCM) is specified in [**RFC3610**]. It allows to pad and authenticate encrypted and unencrypted data. It furthermore requires a nonce for its operation. The size of the nonce is dependent on the number of octets in the length field. In the first 16 bytes of the message, the nonce and the message size is stored. For the encryption itself, CTR is used. It shares the same properties as CTR.

  It allows parallel decryption and encryption as well as random access.

- GCM (Galois Counter Mode)
  GCM has been defined in [**mcgrew2004galois**], and is related to CTR but has some major differences. The nonce is not used (just the counter starting with value 1). To authenticate the encryption, an authentication token $auth$ is hashed with H_{GFmult} and then xor'ed with the first cipher block. All subsequent cipher blocks are xor'ed with the previous result and then hashed again with H_{GFmult}. After the last block the output o is processed as follows: $H_{GFmult}(o \bigoplus (len(A)||len(B))) \bigoplus E^{K^0}(counter_0)$. As a result, GCM is not parallelizable and does not support random access.

  The mode has been analyzed security-wise in **mcgrew2004security** and showed no weaknesses in the analyzed fields [**mcgrew2004security**].

  GCM supports parallel Encryption and decryption. Random access is possible. However, authentication of encryption is not parallelizable. The authentication makes it unsuitable for our purposes. Alternatively, we could use a fixed authentication string.

- XTS (XEX-based tweaked-codebook mode with ciphertext stealing)
  This mode is standardized in IEEE 1619-2007 (soon to be superseded). A rough overview of XTS may be found at [**Martin2010**]. It was developed initially for Disks offering random access and authentication at the same time.

- CMC (CBC-mask-CBC) and EME (ECB-mask-ECB)
  In [**Halevi:2003**] **Halevi:2003** introduces a cipher mode which is extremely costly as it requires two encryptions. CMC is not parallelizable due to the underlying CBC mode, but EME is.

- LRW
  LRW is a tweakable narrow-block cipher mode described in [**tschorsch:translayeranon**]. This mode shares the same properties as EBC but without the weakness of the same clear text block resulting in the same ciphertext. Similarly to XEX, it requires a tweak instead of an IV.

6.4.4.1 Summary of Cipher Modes

Mode \ Criteria	auth	Requires IV	parallelisable	random access
CBC	×	✓	×	×
CCM	×	✓	×	×
CFB	×	✓	✓	✓
CTR	×	✓	✓	✓
ECB	×	×	✓	✓
GCM	✓	✓	×	×
OCB	✓	×[1]	×	×
OFB	×	✓	×	×
PCBC	×	✓	×	×
XTS	×	✓[2]	✓	×
LRW	×	✓[2]	✓	✓
CMC	×	✓[2]	×	×
EME	×	✓[2]	✓	✓

Table 6.1: comparison of encryption modes in terms of the suitability

6.4.5 Padding

A plain text stream may have any length. Since we always encrypt in blocks of a fixed size, we need a mechanism to indicate how many bytes of the last encrypted block may be safely discarded.

Different paddings are used at the end of a cipher stream to indicate how many bytes belong to the decrypted stream.

6.4.5.1 RSAES-PKCS1-v1_5 and RSAES-OAEP

This padding is the older of the paddings standardized for PKCS1. It is basically a prefix of two bytes followed by a padding set of non zero bytes and then terminated by a zero byte and then followed by the message. This patting may give a clue if decryption was successful or not. RSAES-OAEP ist the newer of the two padding standards

6.4.5.2 PKCS7

This padding is the standard used in many places when applying symmetric encryption up to 256 bits key length. The free bytes in the last cipher block indicate the number of bytes being used. This makes this padding very compact. It requires only 1 Byte of functional data at the end of the block. All other bytes are defined but not needed.

6.4.5.3 OAEP with SHA and MGF1 padding

This padding is closely related to RSAES-OAEP padding. The hash size is, however, bigger, and thus, the required space for padding is much higher. OAEP with SHA and MGF1 Padding is used in

[1] included in auth
[2] Requires tweak instead of IV

asymmetric encryption only. Due to its size, it is important to note that the payload in the last block shrinks to $keySizeInBits/8 - 2 - MacSize/4$.

In our approach, we have chosen to allow these four paddings. The allowed sha sizes match the allowed mac sizes chosen above. It is important to note that padding costs space at the end of a stream. Since we are always using one block for signing, we have to take care that the chosen signing mac plus the bytes required for padding do not exceed the key size of the asymmetric encryption. While this usually is not a problem for RSA as there are keys 1024+ Bits required, it is an essential problem for ECC algorithms as there are much shorter keys needed to achieve an equivalent strength compared to RSA.

We have introduced an additional type of padding not related to these paddings. We required for the addRedundancy the following unique properties. Unfortunately, we were unable to find any padding which matched the following properties simultaneously:

- Padding must not leak successful decryption
  For our addRedundancy operation, we required padding that had no detectable structure as a node should not be able to tell whether a removeRedundancy operation did generate content or decoy.
- Padding of more than one block
  Due to the nature of the operation, it is required to be able to pad more than just one block.

Details of this padding are described in the section "Add and Remove Redundancy Operations" in A.

6.5 Routing

If we can follow data from a source to a destination, we may safely assume that the participants of this data exchange are no longer anonymous. So special care should be taken to this aspect. In the past, several approaches have been made to avoid the detection of data while routing. In the following sections, we will look at some basic concepts which have been proposed up until today. We describe their idea and have a look at their weaknesses discovered so far.

In System Implementations, we analyze some related real-world systems regarding how they work and how they have been attacked in the past.

6.5.1 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.

6.5.2 Onion Routing

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 6.6.11.

6.5.3 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).
- 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.

6.5.4 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.

6.5.4.1 DC 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.

6.5.4.2 Annonymous Remailer

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 Heösingius decided to shut it down. For a more in-depth discussion of Pseudonymous Remailers see 6.6.1

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 6.6.3.

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 6.6.4.

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 6.6.5.

6.6 System Implementations

The following sections emphasize on implementations of anonymizing (and related) protocols regardless of their usage in the domain of messaging and anonymity. It is a list of system classes or their specific implementations together with a short analysis of strengths and weaknesses.

Wherever possible, we try to refer to sources. If systems are no longer in use or have never been adopted outside the scientific community, we try to refer to publicly available sources. Some of them do not even have a rudimentary implementation. Instead, they are limited to an idea or a simulator.

If a system shows strong similarities in parts, then we emphasize on these parts and analyze the findings and attacks.

All systems have in common that they are easily identifiable on the network layer except for Tor when working with "pluggable transports'.'

6.6.1 Pseudonymous Remailer

The basic idea of remailers was discussed in [**CHAUM1**]. The most well-known remailer was probably anon.penet.fi, which operated from 1993 to 1996. This type of remailer is often referred to as type-0-remailer.

In principle, an anonymous remailer works as an ordinary forwarding SMTP server. The only difference is that it strips off all header fields except for "from", "to", and "subject" and then replaces the sender and recipient address with pseudonyms respectively with the real address.

This kind of remailer is easily attackable by an authority. Since the remailer knows tuples of pseudonyms and their respective real identities, it was forced in the past to reveal true identities[**penetClosure**]. Furthermore, the message may be monitored at the server or on its way, and then due to the unmodified content matching is easy.

This remailer offers, therefore, no protection against an adversary defined in our problem.

6.6.2 Babel

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.

6.6.3 Cypherpunk-Remailer

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-1-remailers by using encryption for the message. 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 in the example of correlating the message sizes and timing information.

6.6.4 Mixmaster-Remailer

Like Cypherpunk remailers, the Mixmaster remailers were working with onion-like encrypted messages. In contrast to type-1-remailers,

the use of cascading mixes became systematic.

6.6.5 Mixminion-Remailer

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.

6.6.6 Tarzan

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.

6.6.7 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).

6.6.8 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.

6.6.9 SOR (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. While the approach is both simple and effective, it is not suitable against a powerful adversary. First, he 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.

6.6.10 SCION

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.

6.6.11 Tor

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.

6.6.12 I^2P

The name I^2P is a derived from "Invisible Internet Project" according to geti2p.net. 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^2Ps 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^2Ps 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.

6.6.13 Freenet

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 5th 2016 at the time of writing).

6.6.14 Herbivore

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".

6.6.15 Dissent

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.

6.6.16 $\mathcal{P}^5$

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.

6.6.17 Gnutella

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.

6.6.18 Gnutella2

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.

6.6.19 Hordes

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.

6.6.20 Salsa

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.

6.6.21 AP3

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.

6.6.22 Cashmere

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.

6.6.23 Riffle

[**kwon2016riffle**]

6.6.24 Atom

[**kwon2017atom**]

6.6.25 Riposte

[**corrigan2015riposte**]

6.6.26 Pung

[**angel2016unobservable**]

6.6.27 PIR

[**angel2018pir**]

6.6.28 Karaoke

[**lazar2018karaoke**]

6.6.29 Loopix

[**piotrowska2017loopix**]

6.6.30 Stadium

[**tyagi2017stadium**]

6.6.31 Vuvuzela

[**van2015vuvuzela**]

6.6.32 SMTP and Related Client Protocols

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 6.1 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.

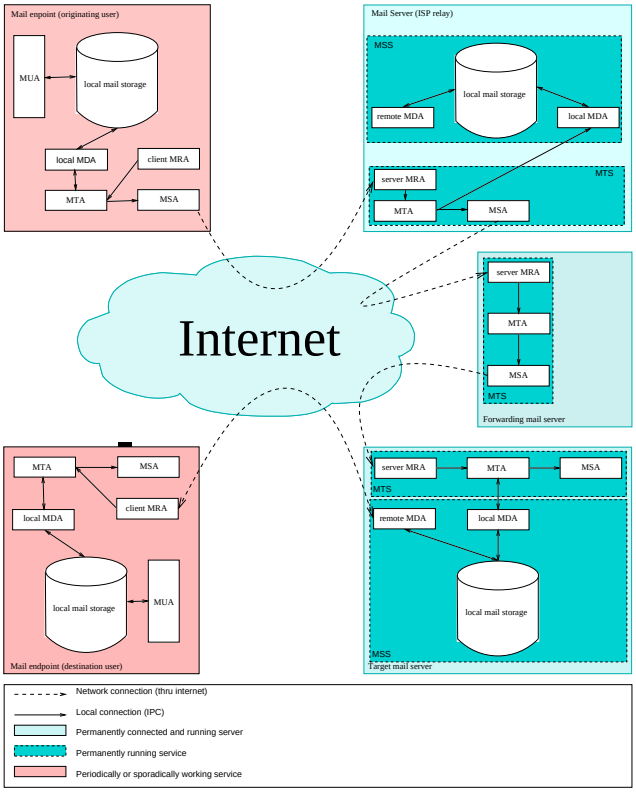

Figure 6.1: Mail Agents

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).

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.

6.6.32.1 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%).

6.6.32.2 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.

6.6.32.3 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.

6.6.33 S/MIME

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.

6.6.34 PGP/MIME

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.

6.6.35 Characteristics of known anonymity Systems

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

6.7 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**].

6.8 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.

6.8.1 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.

6.8.2 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 6.6 or the respective academic works. We list the countermeasures which have been taken to deflect these attacks.

6.8.2.1 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 9.1 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.

	Network Structure						Routing Information		Communication Model					Performance and Deployability				
		Connection		Symmetry							Node Selection							
	Topology	Direction	Synchronization	Roles	Hierarchy	Decentralization	Network view	Updating	Routing Type	Scheduling	Determinism	Selection set	selection probability	Latency	Communication mode	Implementation	Code availability	Context/application
MessageVortex	⊠	⟷	≠	. . .	⋯	○	◐	ϟ	.	≡	∞	◎	◉	H	B	✓	✓	@
Riffle																		
Atom																		
Riposte																		
Pung																		
PIR																		
Karaoke																		
Loopix																		
Stadium																		
Vuvuzela																		

Table 6.2: Classification table for anonymization protocols according to [**Shirazi2018**]

6.8.2.2 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.

6.8.2.3 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.

6.8.2.4 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.

6.8.2.5 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.

6.8.2.6 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.

6.8.3 Denial of Service Attacks

6.8.3.1 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.

6.8.3.2 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.

6.8.3.3 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.

7 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.

7.1 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.

7.1.1 Zero Trust Philosophy

One main disadvantage of almost any system listed in section 6.6 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.*

7.1.2 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.*

7.1.2.1 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.

7.1.2.2 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).*

7.1.2.3 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.

7.1.2.4 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).

7.1.3 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.*

7.1.4 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.*

7.1.5 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.*

7.1.6 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.

7.1.7 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.

7.1.8 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.*

7.2 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 9.4, 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.

7.2.1 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.

7.2.2 Vortex Communication model

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

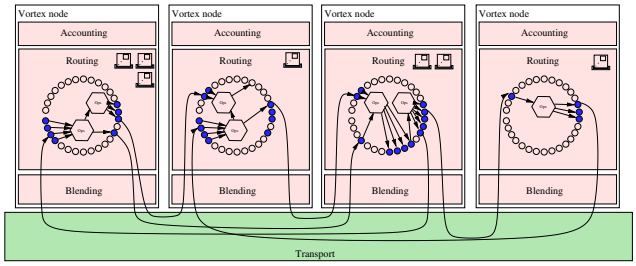

Figure 7.1: A rough protocol outline of the MessageVortex protocol

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.

7.2.3 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.

7.2.4 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.

7.2.5 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.

7.2.5.1 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.

7.2.5.2 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.

7.3 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.

7.3.1 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 7.2. 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.

7.4 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.

7.4.1 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 4.1. 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.

7.4.2 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

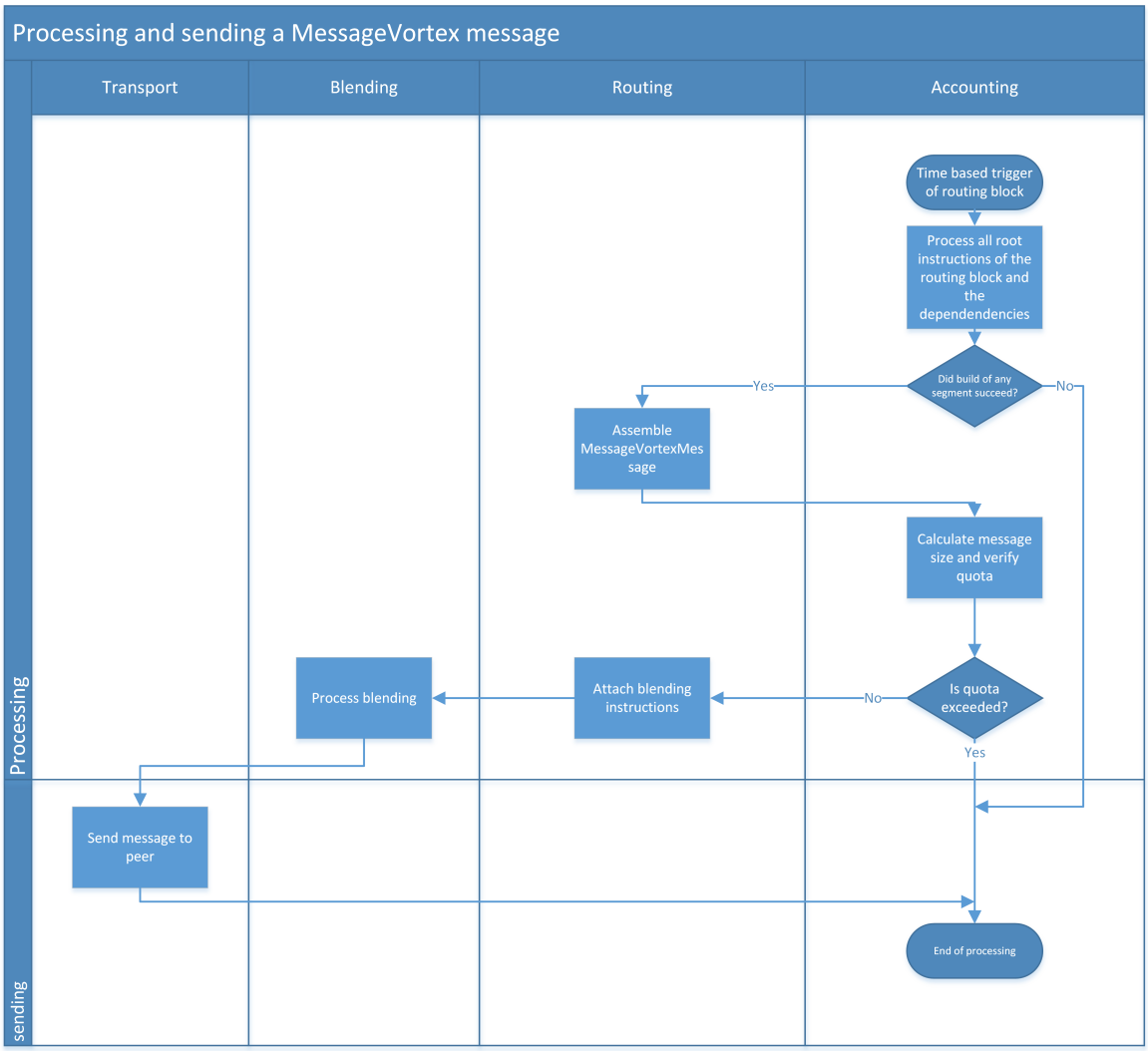

Figure 7.2: flow diagram showing processing of outgoing messages

- 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.

7.4.3 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 III

Results

To verify the hypothesis made in this paper and to analyze the properties of the protocol in a real-world scenario, we implemented a library in Java. This library is capable of handling all message packets and the routing stack as a whole. The implementation is available under `https://messagevortex.net`. The following paragraphs describe the protocol developed in general as a generic approach. Appendix A gives the full ASN.1 representation of the protocol.

As ASN.1 has no means to express encrypted structures, we defined all encrypted fields as `OCTET STRING`. The protocol supports onionized information in an unencrypted form. These unencrypted structures are for debugging purposes only. At no point should this possibility be used in a production environment.

The protocol described in the next chapter is independent of routing. We built a blending layer for SMTP. Layers for other protocols such as XMPP may be defined similarly. We extend the protocol by adding new blending layers for the transport protocol and their addressing schemes.

The Protocol outlined here is the final product and has undergone many development cycles. We dropped a lot of advantageous features and capabilities, such as a mechanism analogous to the SMTP received headers, as they were beneficial but threatened security or anonymity.

8 Vortex Prerequisites

8.1 Hardware

We require no specialized hardware for running Vortex nodes. Instead, we designed Vortex in such a way that ordinary mobile phones may act as Vortex nodes. It is, however, recommended to have a node always connected to the Internet. A mobile phone may disconnect from time to time based on the availability of the network. For our experiments, we used a RaspberryPi Zero W. It is, however, recommended to use a faster, newer model due to the memory requirements of the proof of work algorithm.

The hardware currently requires a network interface and a fully functional JSE VM to run the reference implementation.

8.2 Addresses

A Vortex address is built as follows:

```
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
```

To allow storage of Vortex addresses in standard messaging programs such as Outlook or Thunderbird, we introduced $smtpAlternateSpec$.

The suffix “@localhost” makes sure that any non-participating server does not route a message intended for Vortex. The doubly dotted notation is not RFC compliant but was accepted by all tested client address books. The address is, however, not a valid SMTP address due to its double-dotted notation. We selected this representatiopn to differentiate Vortex addresses from valid email addresses.

The main downside of vortex addresses is that they are no longer readable by a human. The main reason for this is the public key, which is required. We may abstract this further by allowing cleartext requests on the primary email address for the public key. The vortex account must then answer such requests with the valid Vortex address.

The $smtpUrl$ is representing the address in a standard way, which makes it suitable for QR codes and intent filters on Android.

The public key of an address is encoded as follows:

1. The asymmetric key is encoded as specified in the AsymmetricKey in ASN.1
2. The ASN.1 DER representation is then encoded using BASE64

8.3 Transport Layers

As transport layer protocols, we specified the protocols SMTP and XMMP as valid transport layers. In the following sections, we specify the blending properties for these protocols.

8.3.1 Embedding Spec

We always embed VortexMessages as attachments in SMTP and XMPP messages.

The embedding supports some properties. A receiving host chooses the supported properties. We describe valid properties by the blending specification::

```
plainEncoding = "(plain:"<#BytesOfOffset>[,<#BytesOfOffset>]*")
F5Encoding    = "(F5:"<passwordString>[,<PasswordString>]*")"
```

We use mainly plain embedding for our experiments. For better readability, we used a specialized blending layer using unchunked, plain embedding with an offset of 0. The message itself was the ASN.1 block representation of the encoded block. The chosen encoding simplified to see the inner workings of the protocol. For production use, we apply F5 embedding with a generated payload. The current implementation of the blending layer is thus not suitable for production use as the messages remain identifiable or at least suspicious.

8.4 Client

We did not create a Vortex client for sending messages. Instead, we used a standard Thunderbird email client pointing to a local SMTP and IMAP Server provided by a Vortex proxy. On the SMTP side, Vortex does encapsulate where possible mails into a Vortex message and builds an automated route to the recipient. The SMTP part of Vortex may be used to encapsulate all messages automatically with a known Vortex identity into a *VortexMessage*. On the IMAP side, it merges a local Vortex message store with the standard Email repository building a combined view.

Using Vortex like this offers us the advantages of a known client with the anonymity Vortex offers.

Using a proxy has certain downsides. At the moment, the vortex client has only a local store. Such a local store makes it impossible to handle multiple simultaneously connected clients to use Vortex. This limitation is, however, just a lack of the current implementation and not of the protocol itself. We may safely use IMAP storage for storing *VortexMessages* centrally. This statement is true as long as:

- The storage is not identifiable as such.
  This requires:
  - A non-identifiable folder/message structure
  - A storage not identifiable by access patterns
  - The stored messages do have the same strength as the transmitted messages in terms of detectability
- A secured key
  Either the host key is secured sufficiently with KDF, and a passphrase (or similar), or the host key remains off-storage.

8.4.1 Vortex Accounts

By definition, any transport layer address may represent a Vortex identity. This fact may make people believe that their current email

or jabber address is suitable as a Vortex address. This statement is technically perfectly true, but should not be done for the following reasons:

- If an address is identified as a Vortex address, it may be blocked (directly or indirectly) by an adversary. Such blocking would lead to blocking of regular email traffic as well.
- If a vortex node is malfunctioning non-*VortexMessages* should remain unaffected. Isolation is far better if we keep non-Vortex messages in a separate account.
- If a user wants no longer to maintain its Vortex address, he may give up his Vortex transport accounts. If he had been using his normal messaging account for Vortex, he would receive mixing messages which are hard to filter even with a known host key.

8.4.2 Vortex Node Types

8.4.2.1 Public Vortex Node

Public nodes are nodes, which advertise themselves as normal mixes. Just as all nodes, they may be an endpoint or a mix. Typically they accept all requests exactly as outlined in 9.1. As an immediate result of the publicly available information about such a node, the owner may be the target of our censoring adversary. Pressure may be opposed to close down such a node. However, since we do not need a specific account, we may safely close down one transport account and open up a different one. Such account reopenings are even possible on the same infrastructure. We are even able to notify other users of the move and remain reachable, as a user may send a newIdentity request using the old identity.

8.4.2.2 Stealth Vortex Node

This node does not answer any clear-text requests. As an immediate result, the node is only usable by other nodes knowing the public key of this node. The node is, therefore, on a known secret base only reachable. This node type may be used in environments with a censoring adversary. People may form closed routing groups routing and anonymizing themselves. We have to state clearly at this point that putting trust into the routing nodes violates the Zero Trust principle. It is, however, currently the only way to outcurve a censoring adversary. Means such as using distribution lists as endpoints seemed to be of some value at first but turned out just to shift the problem of detection from the routing to the less protected transport layer.

8.4.2.3 Hidden Vortex Node

A hidden node is a special form to a stealth node. It has a predefined set of identities. Only these already known identities are processed. This behavior has certain drawbacks. An existing identity may not be changed, and new ephemeral identities may not be created. As an immediate result, traffic may become pseudonymity. To counter this effect, at least partially, we may use the same local identity for multiple senders. To remove clashes in the workspace, we may use preassigned IDs in the workspace. The sender is only one of all senders knowing the private key of an identity. The advantage of such a node is that identities have unlimited quotas on such nodes, no longer bothering about accounting and refreshing identities. Such behavior seems to be a valuable option when using bulletproof providers.

9 Vortex Protocol Overview

The Protocol details are described in the RFC document in A. The RFC draft contains all the necessary information to build the protocol. The RFC is published through the official IETF channels. Besides the RFC document, additional documents and references may be found on the official website `https://messagevortex.net/`.

The MessageVortex protocol described here is a protocol for asynchronous data transfer. The protocol itself is embedded into a carrier protocol as binary information to avoid easy detection and make it hard to block traffic without blocking other legitimate traffic.

The data transferred is passed through multiple routers. The builder of a routing block (normally the sender) decides upon the following attributes:

- Hops for the message and all decoy traffic.
- Timing behavior of the message.
- Decoy traffic generation.
- Set of possible recipients.
- Set of all nodes involved in routing.

These decisions are compiled into a routing block structure, which is onionized. This routing block may then be used to transfer a message of almost any size. This message is then sent to the involved mixes.

A mix may be just an intermediate station or the final target of a message. Only the recipient of a message can tell whether a message was intended for him or not. Any mix does a certain number of operations on a message. Considering the message, the timing and the operations applied a mix may extract the following pieces of information:

- IP of the sending mix.
- Size of the message received.
- Size of all processed sub-blocks.
- Arrival time of a message.
- Ephemeral identity a message belongs to (an ephemeral pseudonym to the routing block builder (RBB)).
- Validity time of the message on the node.
- Operations applied to the message.
- Size of all blocks sent.
- IPs of the receiving mixes.

A routing node always applies the operations requested in the building instructions to the received data. If this is not done, the message properly may fail to transfer to its final destination.

The operations to be applied to a message are chosen in such a way that they may or may not generate decoy traffic. This design guarantees that valid messages or decoys may not be identified on the operations applied to the message.

Redundancy may be built in a routing block as well as progress indication.

In the taxonomy of [**Shirazi2018**], this protocol would be classified as follows:

- Network topology: full
- Network connection direction: unidirectional
- Network connection synchronization: asynchronous
- Network symmetry roles: hybrid
- Network symmetry topology: flat
- Network symmetry decentralization: fully decentralized
- Routing network view: partial
- Routing updating: Event-based

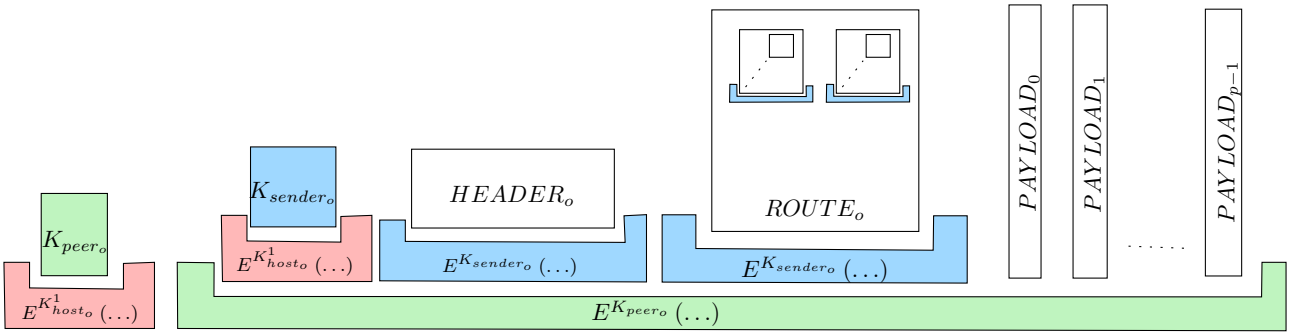

Figure 9.1: Simplified message outline

- Communication routing type: source routed[1]
- Communication scheduling: fair
- Communication node selection determinism: probabilistic
- Communication node selection set: user-based
- Communication node selection probability: uniform
- Performance latency: high
- Performance communication mode: message-based
- Performance implementation: yes
- Performances code availability: yes
- Performance context: email, messaging

9.1 Vortex Message

The outline in figure 9.1 is a simplified view of the message block of MessageVortex.

A Vortex message is a message passed from one router to another. This message may or may not contain any valuable information. The *VortexMessage* is encoded as binary data in the transfer protocol. Every router may decide for himself on the support of algorithms and embedding mechanisms.

The block structure of a Vortex message is as follows:

- Encrypted peer key.
  It contains a symmetrical key for decryption of follow up header information and payload blocks. All symmetric keys are encrypted with a receiving host's public key.
- Inner Message Block (encrypted with peer key)
  - header (encrypted with sender key)
    - Identity
    - "proof of work" information
    - (optionally) header requests
  - Routing blocks (encrypted with sender key)
    - Next hop timing instructions
    - Next hop routing blocks (encrypted)
    - Next hop header
    - Message build instructions.
    - Next hop header key and spec.
    - Next hop blending instructions.
  - Payload blocks

[1] Only partially correct, as the RBB decides on the route. This builder is not necessarily identical to the sender.

It is important to note that there are two symmetrical keys involved in encrypting and decrypting message headers. Having two keys is not a flaw in the protocol but necessary.

The first key of a *VortexMessage* is the message key. This key is only accessible with the private key of the node receiving the message. It allows the decryption of the routing blocks and the header information. The sender of a message block is, therefore, not able to tell if a message contains one or more routing blocks. It is important to note that no other node should have access to this information.

The second key is the sender key located before the encrypted header. The RBB chooses the key. This key protects the inner structure of the Message. It makes it impossible for any node except the sending party or the receiving peer node to detect the inner structure of the message. Without this key, any independent observer with knowledge about the blending capabilities of a receiving node may:

- Easier to identify the block structure.
  This statement remains regardless of whether ASN.1 or length prefixed structures are used. If the structure of a *VortexMessage* can be easily identified, the messages may be logged or dropped.
- Identify the routing block size.
  The value of this information is only minimal as it only reflects the complexity of the remaining routing information indirectly.
- Identify the number of payload blocks and their respective sizes.
  Sizing information is valuable when following the path of a message.

For the exact usage of the keys, see section 9.1.1.

It is important to note that there is no structure dividing the encrypted peer key from the Inner message block. The size of the peer key block is defined by the key and algorithm of the host key.

9.1.1 Key Usage

Several keys are being used during the life of a message. In the following section, we emphasize on the type, the usage, and their specialties.

9.1.1.1 Peer key

The peer key is a message specific symmetrical key known to two adjacent routing nodes. It is generated by the sending routing node and encrypted with the public key of the receiving nodes identity key.

9.1.1.2 Header Key

The header key is a symmetric key protecting the routing and ephemeral identity information of the message. It is prepended to

the header section and protected by the identity key of the processing node.

9.1.1.3 Host Key

The host key is a static, asymmetric keypair existing on a per-user base used to sign messages or encrypt symmetric keys. We refer here as a user any peer participating in the message stream. Every user participating in the Vortex network requires at least one key pair.

Depending on the use-case (e.g., unlikable signatures or scalable security), a user may use multiple key pairs at the same time.

The host key is required for decrypting peer (K_{peer}) and sender (K_{sender}) key.

9.1.1.4 Ephemeral Identity Key

An eID key identifies the sender to a processing host. The ephemeral identity key must be handled in such a way that it is not linkable. It is mainly used to process accounting information.

A user may have multiple key pairs on one routing host.

9.1.2 *VortexMessage* Processing

A node requires FP arithmetic to process messages. To make sure that all implementations on all platforms behave the same, we always use arithmetic as specified in **IEEE754**[**IEEE754**].

9.1.2.1 Receiving Messages

All messages are processed as follows:

1. Extract peer key from the block. A node aborts the operation if a block is invalid or not decryptable.
2. Decrypt sender key with hosts private key.
3. Decrypt header block with the decrypted sender key. Abort if not decryptable or invalid block.
   - Verify identity
   - Check quotas (if any)
   - Extract header key
   - Extract requests (if any)
   - Check replays (if any)
4. Decide if the message should be processed. If not abort here.
5. Decrypt rest of the inner message block with the peer key
6. Extract payload chunks.
7. Decrypt routing blocks with header key.
8. Check $forwardSecrets$ and discard if the inner message block contains any non-matching values.
9. Process instructions

We may split the processing in an authenticated and unauthenticated processing, as shown in figure 9.2. When applying this

Every routing block creates a new message.

The payload of a message is generated according to instructions in the routing block. Timing instructions are relative to the arrival time of the message containing the routing block. This relative timing is necessary as a routing block may be used multiple times (see section 7.2.5.2).

A *VortexMessage* may be composed not earlier than a “validFrom” expressed in the respective routing block.

9.1.2.2 Building and Sending Messages

Any time after a routing block reaches “validFrom” and before the “validTo” is reached, processing of a routing block is triggered. An implementation should, when triggering a routing block for processing, trigger as many routing blocks as possible to make traffic analysis harder.

The message is then built as follows:

1. Check if all building instructions can be fulfilled due to their prerequisites.
2. Build requested payload blocks.
3. Extract peer key from the routing block.
4. Extract headerBlock and encrypted sender key from the routing block.
5. Extract (sender key encrypted) nextHop routing blocks from routing block.
6. Encrypt message using the peer key.
7. Update accounting figures.
8. Blend into the transport layer according to spec and send a message.

9.2 Protocol design

In this section, we emphasize on the protocol blocks. These blocks are extracted from the blending layer and passed to the routing layer. A *VortexMessage* is split into two main parts. The peer key block and the inner message block. The peer key block contains the named key in encrypted form. the inner message block contains the subparts “senderKey”, “header block”, “routing block”, and “payloads”. Although these blocks are described in A as ASN.1 encoded structures, they are not. In the message, they are just subsequent blocks without any structure.

The reason for not using ASN.1 encoding is that it might be possible to identify the unencrypted message on the transport layer as Vortex message due to the ASN.1 structure. By not using this any supporting structure, we make it impossible for an adversary to identify the encrypted structures of a *VortexMessage*.

Without the host key, it is impossible to find any structure within the encrypted message. However, assuming an existing host key detection is easy. The first block must result in an ASN.1 encoded structure containing the symmetric key (and its spec). So for detection of a message with the host key, only the decryption of a single asymmetrically encrypted block is required. A routing node uses the obtained peer key to decrypt the sender key block and the header block. After verification of the header blocks signature, a *VortexNode* has all information required whether full processing is allowed or not. If so, the processing node does decrypt the routing block and check all $forwardSecrets$. After passing this test, all structures are added to the workspace of the eID.

9.2.1 Header block

The header block contains the identity and all information required to decide whether subsequent blocks of the message should be handled.

The header block contains the following data:

- An identity block (identityBlock)
  This block contains data reflecting the identity of the sender and the use of the header and subsequent blocks. This data includes:
  - sending ephemeral identity public key (identityKey)

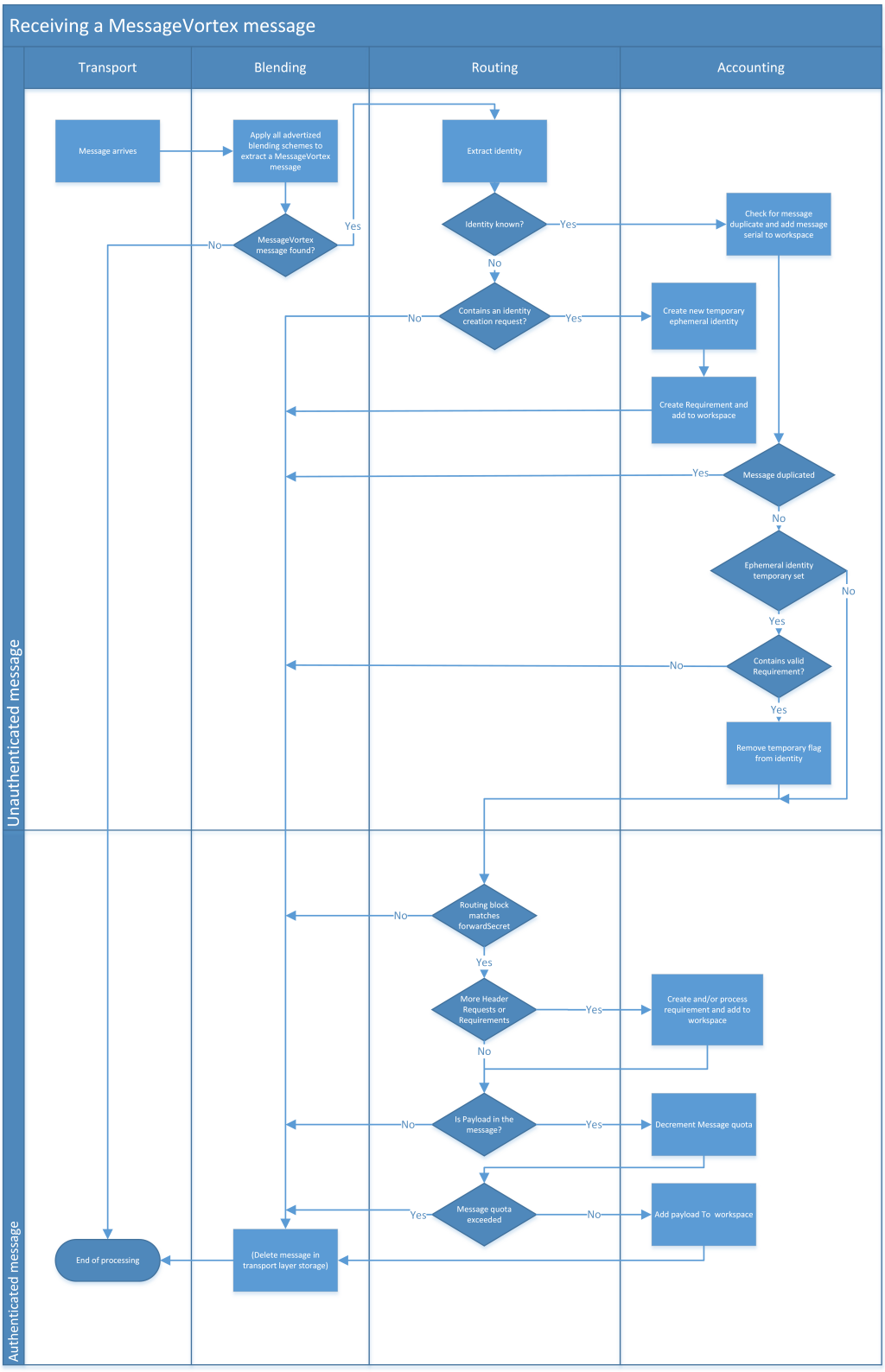

Figure 9.2: flow diagram showing processing of incomming messages

- serial number
- maximum number of replays for the serial for this identity
- Minimum number of seconds for replay protection.
- validity period for this block (in seconds)
- Chain secret for the block ($forwardSecret$)
- Protocol requests to this node
- Identifier and padding for proof of work

- An identity signature (identitySignature)
  Contains a signed hash with the senders private key.

The identity in header blocks is always an ephemeral identity (eID). It exists for a limited amount of time (a low number of days). Creating a new eID is done with an identity request.

The eID identifies the workspace and limits the available routing capabilities. A vortex node only processes known eIDs with sufficient quotas.

9.2.1.1 Requests

Requests are always embedded in a header block. All requests are answered with a provided SURB.

There are several header requests defined:

- newIdentity request:
  This request may be answered with either a reject or a puzzle that is required to solve. Solving the puzzle results in the creation of the identity on the node. The node may reject identities for various reasons:
  - The node is not accepting newIdentity requests
  - The identity is already taken.
  - The identity is not strong enough (it must apply stronger cryptography).
  - The used encryption scheme is not supported by the node.

  An identity may be rejected if the wrong types of keys and key sizes are used. However, it must accept at least the key types and sizes it uses for its own identity.

  If an identity is rejected, the request may not be replayed by the same identity again. A sending party must generate a new identity for a new request.

  This request should be by far the most expensive request. It must, at any time, be more expensive to request a new identity compared to raise the quota of an existing one.

  An identity on this level is always ephemeral and expires after a given period. An eID can not be prolonged for security reasons. Being unable to prolonge the lifetime of a eID has certain drawbacks when using reply blocks. A reply block can only be valid as long as all included identities are valid. To counter this weakness without weakening security, a ctxlessNewIdentity block may be sent to a reply block owner providing him with a new reply block.
- queryPeer request:
  A peer request is a request for publicly known Vortex nodes. This request does offer the possibility of harvesting the Vortex network. We hardened our system, therefore, with the following limitations to make harvesting harder:
  - The request is very costly
  - Only nodes advertising themselves as “public” are disclosed.
  - Only one or two nodes should be disclosed upon request.
  - A node should always pick random nodes out of a 5% pool of known Vortex addresses.

  These measures limit the effectivity of harvesting attacks while giving any node the possibility of bootstrapping itself.
- *queryCapabillity* request:
  This request is the only request answered as a clear-text request. We minimize the possibility of probing by answering such requests only if the node owner agrees to it or generally by public nodes.
- $messageQuota$ request:
  This request raises the number of routing blocks which may be processed for an identity. A node may reject this request depending on the load of the node, personal preferences, or because this identity causes too much traffic.

  It is typically answered for all valid identities only. The node should reject even recently eIDs. A routing node should, however, not send a reply to an unknown eID as this behavior might be used for the probing of a node.
- $transferQuota$ request:
  This request raises the number of bytes that may be transferred for an identity. A node may reject this request depending on the current load, personal preferences, or because this identity causes too much traffic.

  The same restrictions as in $messageQuota$ apply.
- *queryQuota* request:
  This request instructs the node to send information about the given identity.

It is typically answered for all valid identities only. The node may have recently expired. It is, however, not recommended to send a reply to an unknown identity as this behavior might be used for the probing of a node.

9.2.1.2 Replys to Clear Text Requests

It is up to the decision of the node whether it wants to answer a clear-text request or not. Recommended for this behavior is to discard plain text requests.

Discarding such requests should only be a problem when bootstrapping or when adding new identities to their address book.

9.2.2 Routing Blocks

Routing blocks contain an onionised route, chosen by the builder of the routing blocks, and may provide instructions for building subsequent messages.

Routing blocks contain the following information:

- The node specification of the next hop (requires a full identity; may be missing if there is no next hop)
- The moment of processing as a range in seconds since the time of arrival.
- Retention time in seconds since the time of arrival.
- The key blocks for the next hop
- The peer key in plain
- The identity block for the next hop
- The routing block for the next hop
- Payload building instructions

9.2.2.1 Payload Building Instructions

Payload is being built right before sending a block (processing a routing block). The building instructions are built as follows:

$$srcIDs \xrightarrow{\text{build operation}} targetIDs$$

Every node maintains a list of received blocks, including their IDs and building instructions for them. Any node must keep blocks and building instructions during the whole lifetime of a routing block. It may keep it longer. If a conflicting building instruction arrives, all conflicting older rules are removed. Building instructions are always referring to the workspace of the signing eID. It is not possible to reference building instructions of a different identity.

- splitPayload and mergePayload:
  Split a message block into two parts of varying sizes. The size of the first chunk is expressed either absolute or in percent of the original block size.
- encryptPayload and decryptPayload:
  Encrypts and decrypts a payload chunk block with a given symmetric key and algorithm. Please note that this operation changes the size of a message due to the key size and the padding.
- addRedundancy and removeRedundancy:
  Splits a payload block into uniform chunks and adds redundancy information or removes it.

All the operations specified above have in common that they may be applied to decoy traffic as well as on real message data. The size of incoming and outgoing blocks do not relate as messages are increasing the size as well as decreasing in size.

We describe the operations in detail in section 9.2.4.

9.2.2.2 payload Block

The payload block contains the actual message or decoy traffic. Since this block is heavily modified in the course of the transport of the block, it is built simplistically. It contains only the payload data.

An active adversary may always replace a payload block when routing. Any tagging introduced by an active adversary at this point does invalidate the stream. The output after the next hop is entirely unpredictable and, thus, tagging ineffective.

9.2.2.3 Reply Block

Reply blocks are blocks embedded into payload blocks. There are very few reply blocks necessary. Unlike normal data blocks, these messages are not accounted to quotas on the node generating the reply block.

It is up to the node to decide whether it wants to answer a request or not.

Replies are being built as ordinary message blocks. To identify a Vortex message, it must begin with the string "\special" encoded in ASCII followed by a valid reply block structure. No additional bytes may be appended. Blocks with other data should be discarded. To express a block starting with "\special", the token is repeated prefixed with a backslash.

- *replyCapability* block:
  The reply contains the following information:
  - Supported Vortex transports, including blending specification.
  - Maximum quota.
  - Supported ciphers and hashes.
  - Maximum number of simultaneous valid header serials.
  - Maximum number of simultaneous valid building operations.
  - Maximum identity lifespan in seconds.

  It lists the capabilities a node advertises to the public.
- *requirement* block:
  Requirement blocks denote a requirement a requester has to fulfill before a previously sent request is processed. Usually, proof of work puzzles needs to be solved to allow a request to be processed. Alternatively, a commercial supplier may request payment in digital currency. Currently, supported digital currencies are Bitcoin, Ethereum, and ZCash.
- *replyStatus* block:
  General answer block, which is signaling a status. The block is limited in length to minimize the misuse of bandwidth. The Block contains the following data:
  - Three digit status number
  - Sending node identity
  - Status text (optional)
  - Affected block ID (optional)
- *ctxlessNewidentity* block:
  This block may be used to signal the change of identity to a recipient. As this request is signed with the old known identity, no means should exist to hijack such an identity.

  This request contains:
  - old Identity
  - new Identity
  - Signature (with old identity)

  This message may arrive at any time. Any recipient might decide on its own whether it wants to accept the update or not.

  A node should not accept an identity update if the strength of the new identity has been lowered compared to the old identity. A client may make a difference in the fact of whether the transport layer address or the key is exchanged.

9.2.3 Accounting

Accounting covers several purposes in this system:

- It makes the system costly for nodes sending bulk messages.
- It protects from replaying.
- It offers an eID with pseudonymous characteristics and a limited lifespan.

As accounting data may be used to overfill a nodes accounting tables, special care has been taken to limit the number of information which has to be maintained per identity. We furthermore tried to minimize the risk that someone might occupy the accounting memory of a node without costs. Moreover, any node may cancel an illicit behaving identity at any time.

It is essential that the accounting described here is for routing nodes. A node building a routing block requires more accounting information as it has to keep track of all ephemeral identities.

9.2.3.1 Accounting Data of an Ephemeral Identity

For an ephemeral identity, very little information has to be kept. This identity expires after a certain amount of time. The maximum time may be queried with a capability request. The choice of encryption type and key size is left to the node requesting the identity. However, a node may reject the request if it considers the identity to be unsafe, it has no more capacity for new identities, or if it would create an identity clash on the current node.

The following data has to be kept per identity:

- $\mathbf{EID}\langle expiry, pubKey, mesgsLeft, bytesLeft, temporary\rangle$
  The **EID** tuple is the longest living tuple. It reflects an ephemeral identity and exists as long as the current identity is valid. All other tuples are short living lists. As the server decides if he accepts new identities or not, the size of this data is controllable. The temporary flag describes an identity which has an unsolved puzzle.
- $\mathbf{Puzz}[]\langle expiry, request, puzzle\rangle$
  The array **Puzz**[] is a list of not yet solved puzzles of this eID. Every puzzle has a relatively short lifespan (typically below 1d). A routing node controls the size of this list by only accepting requests to a certain extent. Typically this list should not surpass two entries as we should have either a maximum of two quota requests or one identity creation request open.
- $\mathbf{Replay}[]\langle expiry, serial, numberOfUsages\rangle$
  The array **Replay**[] is a list of replayable MURBs. List entries are created upon their first usage and remain active until the block is expired.

The following data has to be kept for routing within the eIDs workspace:

- $\mathbf{Build}[]\langle expiry, buildOperation\rangle$
  The array **Build**[] is a list of building instructions for a message. The server may decide at any time to reject a too big list or long-living message. Thus he may control the size of this list as well. However, controlling the size of this list will most likely result in the non-delivery of a message.

- **Payload**[]$\langle expiry, payload, id \rangle$
  The array $Payload[]$ reflects a list of all currently active payloads. Servers may decide to store derivatives of payloads. However, as derived payloads inherit their expiry from the generating operation, such behavior may be safely omitted and operations executed if their result is required.

All items have an expiry time, and no expiry time may surpass the expiration of the eID.

9.2.4 *VortexMessage* Operations

All operations are expressed as described in section 9.2.2.1. The following sections give important details about the implementation of the operations.

9.2.4.1 SplitPayload Operation

The splitPayload operation splits a payload block into two chunks of different or equal sizes. The parameters for this operation are:

- source payload block pb_1
- fraction f
  A floating-point number which is describing the size of the first chunk. If the fraction is “1.0”, then the whole payload is transferred to the second target chunk

If $len(pb_1)$ expresses the size of a payloadblock called pb_1 in bytes then the two resulting blocks of the SpitPayload Operation pb_2 and pb_3 have to follow the following rules:

$$split(f, pb_1) = \langle pb_1, pb_2 \rangle \quad (9.1)$$
$$pb_1.startsWith(pb_2) \quad (9.2)$$
$$pb_1.endsWith(pb_3) \quad (9.3)$$
$$len(pb_2) = floor(len(pb_1) \cdot f) \quad (9.4)$$
$$len(pb_1) = len(pb_2) + len(pb_3) \quad (9.5)$$

9.2.4.2 MergePayload Operation

The mergePayload operation combines two payload blocks into one. The parameters for this operation are:

- first source payload block pb_1
- second source payload block pb_2

If $len(pb)$ expresses the size of a payloadblock called pb in bytes then resulting block of the MergePayload Operation pb_3 have to follow the following rules:

$$merge(pb_1, pb_2) = pb_3 \quad (9.6)$$
$$pb_3.startsWith(pb_1) \quad (9.7)$$
$$pb_3.endsWith(pb_2) \quad (9.8)$$
$$len(pb_3) = len(pb_1) + len(pb_2) \quad (9.9)$$

9.2.4.3 EncryptPayload Operation

The encryptPayload operation encrypts a payload block pb_1 symmetrically resulting in a block pb_2. The length of block pb_2 may vary according to mode and padding chosen. The parameters for this operation are:

- Source payload block pb_1
- Encryption specification $spec$
- Symmetric key k

The operation follows the following rules (please note section 2.1 for notation):

$$encrypt(pb_1, spec, k) = pb_2 \quad (9.10)$$
$$pb_2 = E^{K_a}_{spec}(pb_1) \quad (9.11)$$
$$len(pb_2) \geq len(pb_1) \quad (9.12)$$

9.2.4.4 DecryptPayload Operation

The decryptPayload operation decrypts a payload block pb_1 symmetrically resulting in a block pb_2. The length of block pb_2 may vary according to mode and padding chosen. The parameters for this operation are:

- Source payload block pb_1
- Decryption specification $spec$
- Symmetric key k

The operation follows the following rules (please note section 2.1 for notation):

$$decrypt(pb_1, spec, k) = pb_2 \quad (9.13)$$
$$pb_2 = D^{K_a}_{spec}(pb_1) \quad (9.14)$$
$$len(pb_2) \leq len(pb_1) \quad (9.15)$$

9.2.4.5 addRedundancy and removeRedundancy Operation

These operations build the core of the routing capabilities of a node. The operation allows a RBB to add to a message redundancy information or to rebuild a block from a chosen set of information.

The Operation itself is shown in figure 9.3.

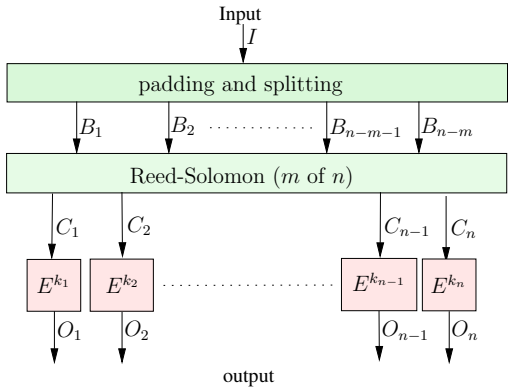

Figure 9.3: Outline of the addRedundancy operation

It may be subdivided into the following operations:

- Pad the original message block in such a way, that all resulting blocks are a multiple of the block size of the encrypting cipher.
- Apply a Reed Solomon operation in a given GF space with a vanderMonde matrix.
- Encrypt all resulting blocks with unpadded, symmetrical encryption.

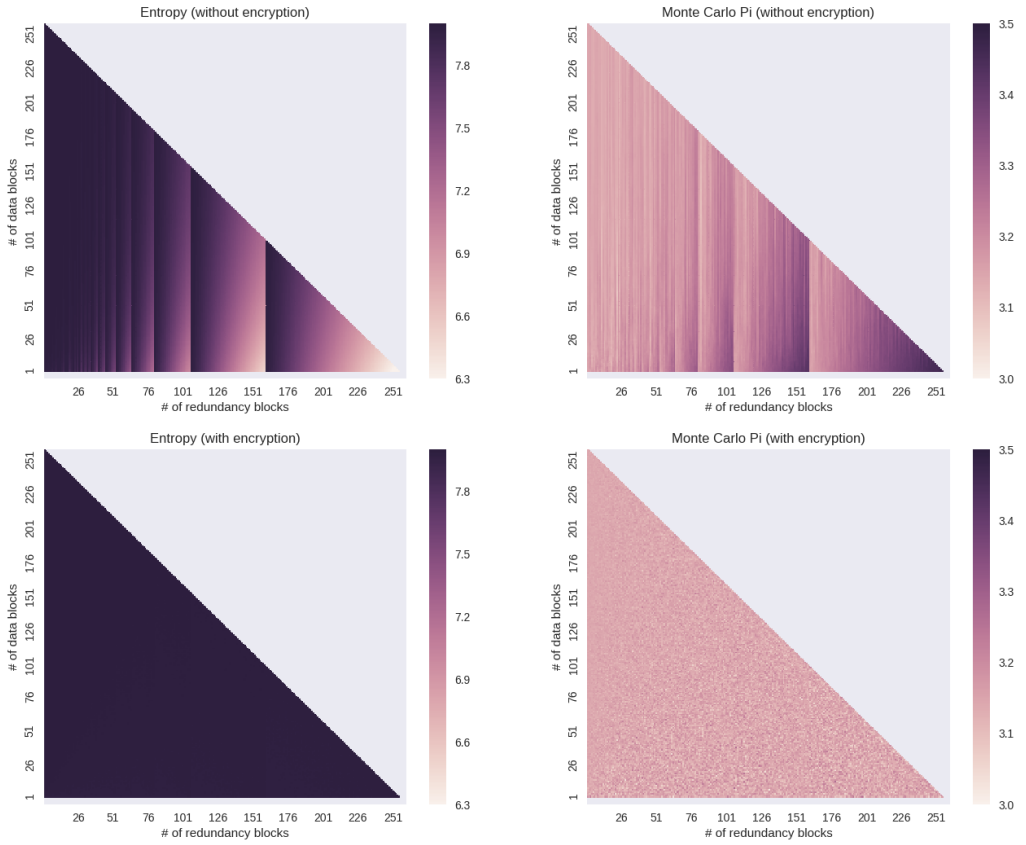

Figure 9.4: Resulting entropy of addRedundancy with and without encryption step

The padding applied in the first step is non-standard padding. The reason for this lies in the properties required by the operation. The presence of standard padding may leak, whether the block has been successfully decrypted or not. Therefore, we created a padding with the following properties:

- The padding must not leak whether the rebuild cycle of the operation was successful or not.
- Anyone knowing the routing block content and the transmitted message must be able to predict any treated block, including all padding bytes.
- The padded content must provide resulting blocks of required size to enable non-padded encryption after the RS operation
- The padding must work with any size of padding space.
- The padded and encrypted block must not leak an estimate of the original content.

The padded block $\mathbf{X}$ is created from a padding value p, the unpadded block $\mathbf{M}$ and a series of padding bytes. We build $\mathbf{X}$ for a function $RS_{\text{m of n}}$ and an encryption block $\mathbf{M}$ sized K as follows:

$$i = len(\mathbf{M}) \tag{9.16}$$

$$e = k \cdot n \tag{9.17}$$

$$l = \left\lceil \frac{i+4+C2}{e} \right\rceil \cdot e \tag{9.18}$$

$$p = i + \left(C1 \cdot l \pmod{\left\lfloor \frac{2^{32}-i}{l} \right\rfloor \cdot l} \right) \tag{9.19}$$

$$\mathbf{X} = \langle p, \mathbf{M}, R_t(s, l-i-4) \rangle \tag{9.20}$$

The remainder of the input block, up to length l, is padded with random data. The random padding data may be specified by RBB though a PRNG spec t and an initial seed value s. The message is padded up to size L. All resulting, encrypted blocks do not require any padding. This baecause the initial padding guarantees that all resulting blocks are dividable by the block size of the encrypting function. If not provided by an RBB, an additional parameter $C1$ is chosen as random positive integer and $C2 = 0$ by the node executing the operation.

To reverse a successful message recovery information the of a padded block $\mathbf{X}$, we calculate the original message size by extracting p and doing $len(\mathbf{M}) = p \pmod{len(\mathbf{X})}$.

This padding has many important advantages:

1. The padding does not leak if the rebuilding of the original message was successful. Any value in the padding may reflect a valid value.
2. Since we have a value $C2$, the statement that a message size is within $len(\mathbf{X}) < size < (len(\mathbf{X}) - k \cdot n)$ is no longer true and any value smaller $len(\mathbf{X}) - k \cdot n$ may be correct as well.
3. An RBB may predict the exact binary image of the padded message when specifying $C1$, $C2$, and $R_t(s,)$.

After the padding, the date is ready for the *addRedundancy* operation. We first group the data vector into a matrix $\mathbf{A}$ with m columns to do the operations efficiently. The previous padding guarantees that all columns have a length, which is dividable by the block size of the encryption step applied latter.

$$t = n - 1 \tag{9.21}$$

$$\mathbf{A} = vec2mat\left(\mathbf{X}, \frac{len\,(\mathbf{X})}{m}\right) \tag{9.22}$$

$$\mathbf{V} = \begin{pmatrix} 0^0 & 0^1 & 0^2 & \cdots & 0^{(m-1)} \\ 1^0 & 1^1 & 1^2 & \cdots & 1^{(m-1)} \\ 2^0 & 2^1 & 2^2 & \cdots & 2^{(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t^0 & t^1 & t^2 & \cdots & t^{(m-1)} \end{pmatrix} \tag{9.23}$$

$$\mathbf{P} = \mathbf{VA}\,(GF\,(2^\omega)) \tag{9.24}$$

$$\langle \mathbf{Q_1}, \ldots, \mathbf{Q_n} \rangle = row2vec(P) \tag{9.25}$$

$$R_i = E^{K_i}\,(Q_i) \tag{9.26}$$

We do the Reed-Solomon operation by employing a Vandermonde matrix ($\mathbf{V}$). We build the data matrix ($\mathbf{A}$) by distributing the data into $\frac{len(\mathbf{X})}{m}$ columns. This results in a matrix with m rows. Unlike in error-correcting systems, we do not normalize the matrix so that the result of the first blocks is equivalent to the original message. Instead, the error-correcting information is distributed over all resulting blocks ($\mathbf{Q_i}$). Since the entropy of the resulting blocks is lowered as shown in figure 9.4 and may thus leak an estimate of how a resulting block may have been treated, we added the encryption step to equalize entropy again. The previously introduced padding guarantees that there is no further padding on block-level required. The key used to encrypt the single blocks must not be equivalent. Equivalent keys have the side effect encrypting equal blocks into the same cypher-text. We observed faint but statistically relevant reminders of the unencrypted graphs when treating the same block with the same key and different redundancy parameters.

9.3 Request Processing

VortexMessage requests allow a Vortex node to gain knowledge about the Vortex network and create new identities. A request may contain either a request for information such as the current quota or capabilities of the router. It may contain a request for a new identity, or it may contain a request for raising the quotas. The following sections explain the different kinds of requests.

9.3.1 Requests

These requests are contained in the header portion of a Vortex message. These requests are purely for bootstrapping and maintaining the quota system and for requesting network capability.

The request information is defined in section 9.2.1.1. For more information about the exact binary representation of all blocks and data, see chapter 9.4.

Any node decides on its own what type of requests are being answered.

A node not replying to clear text request is called a “stealth node” (see 8.4.2.2). Such a stealth node discloses itself only to participants who do already know at least the public key of the node. This usually means that they have “earned” this information by issuing a queryPeer request to another node, and obtaining the information did already generate costs to the sender.

A node only replying to a fixed set of identities (in that specific case they are not ephemeral) is called a “hidden node” (see section 8.4.2.3).

It is recommended that unencrypted requests are not answered. A node may decide to answer unencrypted queryCapability requests to enable clients to bootstrap without (or with a minimal) network knowledge.

If a message contains n requests in a header block, it must supply at one reply blocks at the beginning of the routing block list. All reply blocks are concatenated and sent using the reply block. If the first block in the routing block list is not a reply block, the request will fail.

9.3.1.1 QueryCapability Request

This request is primarily used to initialize a conversation with a node. It contains valuable information about the capability of the node as well as information about the embedding supported or the encryption. A node may or may not reply to queryCapability requests. Doing so confirms the node to be participating in the vortex network.

The only valid reply is a replyCapability message in encrypted form.

9.3.1.2 NewIdentity Request

If this request is accepted, it generates a new ephemeral identity. The identity itself is stored in the header fields. The standard behavior is to reply with a replyPuzzleRequired block.

This request generates a temporary, ephemeral identity for the limited time denoted in the validity field of the reply block. No quota is assigned during that phase. As soon as the identity offers a correct puzzle solution, the requested quota is allocated, and the identity may be used for subsequent requests.

If a puzzle is not solved within the given time, the temporary identity may be deleted. A node should not accept a solved puzzle after the given duration.

Requests with a too high message or transfer quota should not be answered with a replyPuzzleRequired block containing a zero-length puzzle.

9.3.1.3 QueryPeer Request

As outlined in section 9.2.1.1, this request should be very costly, and the harvesting of public addresses should be hard. Furthermore, this request should always disclose the nodes from a fixed subset of the nodes known to the queried mix. This maximizes the effort to harvest participating nodes.

It is at the same time worth mentioning that this limit may oppose a thread that traffic is concentrated on similar nodes within participating nodes using the same initial set of public addresses to bootstrap. This because if using the same node to bootstrap for multiple participants within a group results in peer knowledge, which is similar. They are thus resulting in a similar network.

Participants belonging to more than one group will evolve as their different peer partners will result in different anonymity sets over time, even when applying precisely the same reproducible rules.

9.3.1.4 TransferQuota, MessageQuota, and QueryQuota Request

This request may be accepted by a node if and only if the sender is a valid identity. It may be accepted even for a temporary identity.

The transfer quota offers the capability to raise the number of bytes an identity may transfer. This quota is increased upon request by the ephemeral identity at any time. It is up to the owner of an ephemeral identity and the node using the identity to decide whether an identity may be kept or not, respectively, its quotas raised.

The Message Quota is a quota not limiting the number of bytes but the number of messages. As every message generates accounting overhead, this number has to be limited as well. There are constraints similar to transferQuota when raising this value.

The queryQuota request enables the owner of an ephemeral identity to query the current amount of remaining messages, respectively, bytes.

9.3.2 Reply Blocks

Reply blocks are as outlined in section 9.2.2.3 prefixed payload blocks.

9.3.2.1 ReplyCapability block

The replyCapabilityBlock is the reply block to a queryCapability Request. The information provided here is outlined in section 9.2.2.3. It is important to note that this block is even when requested in plain is always onionized and thus unreadable for third parties.

A node may offer different capabilities to known identities than to anonymous clear-text requests.

9.3.2.2 replyPuzzleRequired block

The replyPuzzleRequired block is the block reflecting the payment for a requested operation such as newIdentity, queryPeer, transferQuota, or messageQuota request.

Every puzzle block will create an accounting entry, as outlined in section 9.2.3.

A node may reject an operation for any reason, including exceeding a high amount of outstanding puzzles.

A reply containing a null length puzzle means that the requested operation is rejected.

9.4 Protocol Usage

This approach is different from all approaches discussed previously. Unlike them, we put complete distrust into the infrastructure being used. Furthermore, we do not rely on a custom server infrastructure on the Internet. Instead, we take advantage of the availability of Internet-connected devices such as mobile phones, tablets, or even commonly available SoC such as RaspberryPi or similar. It is still tough to maintain a server on the Internet. Considering the vastly growing amount of automated attacks carried out against Internet-connected servers, it is not advisable or realistic to assume that a future user of this system owns either a server or connects to a service that is offering anonymizing services. These infrastructures would be susceptible to monitoring or even banning. Instead, we take a different approach.

We use common messaging protocols as transport layers and connect to them using the respective client protocols. The actual mixes are operated by the users on their "always connected" devices. Such a system is far less reliable than a traditionally run server as this hardware is typically cheap and generally connected to the Internet using a bandwidth shared media.

The basic idea is that a client generates all traffic (including decoy and diagnosis) by itself. It defines the routes a message takes through the mixes and decides which targets are receiving dummy traffic at the same time. In such a system, even when possessing all the nodes routing the traffic (without the endpoints), an anonymity set of k (whereas the sender defines the size of k) is guaranteed.

As decoy traffic is generated with the same operations as the real content is split, it is impossible for an adversary running a node to determine whether he is generating noise or processing the real message. All nodes, regardless of endpoint or mix, implement the same logic, and fulfill the same functions, which make it impossible to determine the function. Exit nodes, as in Tor or similar systems, do not exist.

9.5 Accounting

The Accounting layer maintains all local identities and controls the overall load to the system. He processes requests from an ephemeral identity and generates replies to these requests.

In table 9.1, we show under what circumstances a reply to a header request should be sent. The capitalized words MAY, MUST, SHOULD, and SHOULD NOT are used as defined in RFC2119[**RFC2119**].

9.6 Routing

The routing of a message is simple. A workspace of an eID contains routing blocks and payload blocks. A routing block has an active time window. Anytime during that time window, a routing layer processes the routing instructions contained in the assembly operations of the routing block. If successful, the message will be sent using the specified blending layer and target address.

9.7 Blending layer

The blending layer must be crafted carefully. A blending layer is responsible for the sensible content generation of the transport media plus embedding a *VortexMessage* into the transport layer according to specs provided by the original sender.

The original sender has no control over the plain text messages to avoid the possibility of sending targeted messages over the transport layer using MessageVortex. This differentiates MessageVortex from other systems having "exit nodes" such as Tor.

9.7.1 Plain Inclusion

The data stream has of the MessageVortex protocol has no structure visible from the outside. This property allows embedding as structureless information in files with a similar entropy. We did an analysis of common file formats on the Internet to figure out which file format is suitable for this type of inclusion.

It is essential to understand that this is a frail form of information hiding. A human observer may easily tell "real" data from "broken" data apart. A human is not able to tell whether a file is severely broken or contains a *VortexMessage*.

9.7.1.1 File Type Candidates

We were unable to find any scientifical data regarding what type of traffic or attachment is common on the Internet. We, therefore, tried to analyze mail logs (SMTP) of a mail provider. We were scanning 567594 emails for attachment properties after the spam elimination queue. 16.5% of all scanned messages had an attachment. The top 20 attachment types distributions are shown in table 9.2.

Type	%
image/jpeg	27.4
application/ms-tnef	13.7
image/png	13.3
application/pdf	10.7
image/gif	7.4
application/x-pkcs7-signature	5.4
message/rfc822	7.0
application/msword	3.1
application/octet-stream	3.0
application/pkcs7-signature	2.3
application/vnd.....wordprocessingml.document	1.4
message/disposition-notification	1.1
application/vnd.ms-excel	0.8
application/vnd.....spreadsheetml.sheet	0.6
application/zip	0.5
application/x-zip-compressed	0.5
image/pjpeg	0.4
application/pkcs7-mime	0.4
video/mp4	0.4
text/calendar	0.4

Table 9.2: Distribution of top 20 attachment types

As expected, the number of images within mail was very high ($\approx 50\%$). Unfortunately, we were unable to analyze the content of ms-tnef attachments retrospectively. It seems that based on these

Request \ Criteria	unknown identity; cleartext	unknown identity; encrypted	expired identity; encrypted	known identity; encrypted
newIdentity	SHOULD NOT	MAY	Invalid (Error)	Invalid (Error)
queryPeer	MUST NOT	MUST NOT	MAY	MAY
queryCapability	SHOULD NOT	MAY	MAY	MUST
messageQuota	MUST NOT	MUST NOT	MAY	MUST
transferQuota	MUST NOT	MUST NOT	MAY	MUST

Table 9.1: Requests and the applicable criteria for replies

figures, information hiding within images in email traffic is a good choice.

For our implementation, we worked with F5 blending into jpeg images, as this choice seemed to undermine credible content based on table 9.2.

9.8 Considerations for Building Messages

In a worst-case scenario, we assume that an adversary is controlling most of the network utilized for anonymization. While this is not necessarily a problem, it allows an adversary to track a message while agents are being used under his control. So for simplicity and as a worst-case assumption, we always assume that an adversary has perfect knowledge of an associated message flow. This is, however, a worst-case scenario. One missing agent disconnects the whole chain, and as messages are no longer traceable.

9.8.1 Ephemeral identities

Any *VortexMessage* sender may maintain one or more ephemeral identities per node. These identities might be active in parallel, overlapping, or even with interruptions. A routing block building node is advised to select multiple trustworthy nodes (such as known endpoints) and add some publicly available nodes or nodes obtained by bootstrapping. Those nodes are not trustworthy and may be chosen from a list of different networks.

9.8.2 Timing of messages

Messages are flowing in a timed manner through the network. As a RBB has to take into account that potential routing mechanisms of the transport layer consume time, a message is delayed in each hop. The RBB controls the timing and duration of delivery. Depending on the number of hops for the longest path of the message and the delay windows on every hop, the total message has a delay, which is controllable by the RBB.

9.8.3 Diagnostics

To diagnose the flow of a message, any part of the message may be sent directly or indirectly back to the RBB. This allows him to judge upon the message progress and whether nodes are well-behaving or not.

There is no fingerprinting operation available for making such a diagnosis. Such operations would make the traffic identifiable as diagnostic traffic.

9.8.3.1 Implicit Diagnostic

When a message contains a routing block sending any parts back to the RBB, we call this implicit diagnostic. Any block built by the addRedundancy function may be seen as a kind of fingerprint over the whole message. A block sent back to the originating node may, therefore, reflect the message state up to this point and the way back.

9.8.3.2 On-Demand Diagnostic

Whenever a message fails or suspected fails, a new routing block may be composed picking up parts of the message in workspaces anywhere within the participating nodes. This kind of diagnostic we call On-Demand diagnostic.

On-demand diagnostic allows error conditions identified by implicit diagnostic to be tracked and narrowed down to the first offending node.

9.9 Considerations for Routing Messages

Messages should always be sent nearby other messages timewise. This means that the best moment for sending a message in a ready queue is at a time when sending other messages is due. However, no optimization should be done to send as many messages as possible at the same time. This would lead to a foreseeable behavior of the routing layer and thus to miss-usable behavior.

The approach is furthermore heavily dependent on the transport protocol and builds on top of a new obfuscating/routing layer. For this system to become a real peer-to-peer approach, some additional quirks are required. A message-Vortex-Account always needs an active routing handler. This routing handler may be introduced by new server capabilities or by having a device handling the routing from the client-side. For this reason, we built a RaspberryPi appliance capable of connecting to one (or more) accounts fetching incoming emails, analyzing them, and reroute them if necessary. Although the system is designed to be run on a RaspberryPi, the software might be installed to any Java-capable client. The RaspberryPi is just one affordable, lightweight device that offers all required capabilities.

There was up until very late a routing log functionality in the protocol. This functionality did, however, have the disadvantage that it allowed bugging and could disclose intermediate mixes to a recipient who did not comply with the policy the mixes might have chosen. Therefore this feature was dropped and replaced with the fetch block behavior.

The RBB controls the blending. Besides that, he has no control over it. If blending is done carelessly, a message can be easily detected and thus disrupted.

The message leaks its size when a routing block is reused. This is due to the version number and the ephemeral identity contained in the header. The message chunks reflect approximately the message size compared to the previous message sent.

10 Verification of requirements

In the previous sections, we identified a list of requirements.

In the following subsections, we will iterate through all requirements and verify to what degree we achieved the goal.

RQ1 (Zero Trust): We have not put any trust in an external infrastructure. While we do assume that all routing nodes act as defined. A misbehaving node may be identified and eliminated without putting any trust in other nodes. Analysis has shown no means for a misbehaving node, which might be intentional or unintentional endangering anonymity at any time. We do not rely on any third-party technology or infrastructure for our anonymity.

This requirement is, therefore, fulfilled.

RQ2 (Equal nodes): No node has additional privileges or offers additional services. All are equal and share the same privileges.

This requirement is, therefore, fulfilled.

RQ3 (Undetectable): Detectability is highly dependent on the respective implementation of the blending layer. We can clearly state that research in F5 showed no weaknesses so far. Plain blending is not suitable as human censors may detect such blending.

We did not invest any research effort in the clear text part of the messages. We can clearly state that messages do not have to look like human-generated messages. Status messages from monitoring systems or similar M2H messages are suitable as well. Such messages explain uncommon activities such as 24x7 operation or systematic message outlines. For the academic implementation to be of any value for the public, the current implementation has to be replaced by a blending layer creating suitable content in this respect.

This requirement can be fulfilled, but the current implementation does not meet the required criteria.

RQ4 (untagable): Messages may not be tagged. All content is either strictly onionised or defined and linked with unknown hooks. Tampering with a message will typically cause the message delivery to fail at the next node. Furthermore, may tampering be detected.

This requirement is, therefore, fulfilled.

RQ5 (unbugable): There are always means to bug a message. As we put trust in the sender and recipient, and we know already that an intermediate mixing node is not able to modify the message, the protocol is hard to bug. There may be a possibility to bug a message with a routing log entry over DNS. If a recipient is not resolving names or trusts in the content of such a message, he is safe.

This requirement is, therefore, fulfilled. It may be only partially fulfilled if log entries are not handled with care.

RQ6 (replay): Messages may only be replayed a limited amount of times. The number of replays is controlled by the sender and may not be altered by any mix. A malfunctioning mix replaying more often than allowed will not be able to extract any information than the information it obtained when sending the first time.

This requirement is, therefore, fulfilled.

RQ7 (accounting): All identities generated are not traceable as any identity is generated without any context and may not be mapped to an older or newer identity (perfect anonymous forward identity). Neither the source nor the replies may be used to be traced as all messages.

This requirement is, therefore, fulfilled.

RQ8 (anonymisation): Anonymity is hard to proof.

The following statements are the findings of the previous chapters:

- Routing nodes are identifiable by an in-depth inspection.
  A routing node features unique features that can be identified.

  Identifyable properties discovered are:

  - The usage of the plain embedding is identifiable by a human and using probabilistic approaches even by a scanner.
  - A router is always connected and sends messages at any time of day
  - A router is connected to multiple accounts
- A routing node can link a message to an ephemeral identity
  This is a minor issue and is countered by the fact that ephemeral identities have a very short lifespan and are unconnected.
- A routing node learns about other nodes over time.
  A routing node is unable to communicate with peers without a host key. It may, however, learn the endpoint address of its peer. Assuming a censoring adversary, this may be a problem in a single case as the provider of the transport layer may be forced to block the account.

On the other hand, no node can tell by observing traffic if another node is a final recipient or just another router.

There are, however, some weaknesses in the protocol. As the implementation is currently connecting simultaneously to the transport layer endpoint (email or Jabber account in the current implementation) and the Vortex account, that fact might identify the user. Using an anonymization proxy could solve the problem, but it would violate the Zero trust principle.

A sender is capable of leaking the presence of a receiver to a global observer.

This requirement is, therefore, only partially fulfilled. However, the weakness is very faint.

RQ9 (bootstrapping): The header request peer functionality allows to query for routing nodes. The key handling of the protocol allows using a node without disclosing its host key. Each node may decide on whether it leaks its own identity.

This requirement is, therefore, fulfilled.

RQ10 (algorithmic variety): The protocol lists at least two completely independent algorithms of each kind to be supported. This allows switching if an algorithm has been broken. Wherever possible, a well-known algorithm and an algorithm basing on an entirely different mathematical problem have been chosen.

This requirement is, therefore, fulfilled.

RQ11 (easy handleable): The protocol allows the use of clients already available and know to the user to send messages. The whole encryption and anonymity problem is hidden in a local proxy. This allows users to stick to their favorite tools.

This requirement is, therefore, fulfilled.

11 Security Analysis

In the following sections, we emphasize on attacks targeting either sender-recipient tuples or participating nodes.

Based on the threat model, we may safely assume the following key points:

- An adversary knows and controls a significant number of nodes.
- An adversary may observe the traffic at any point without getting any information about the message content
- An adversary is not capable of matching multiple messages on different nodes to one message.

We always assume an adversary to have more knowledge than we think he may extract from the messages.

- We assume that an adversary knows all messages of a transaction running over his nodes and matches them correctly to the same message.
- We assume that an adversary is aware of any message only containing decoy traffic.

We assume that the adversary is targeting the following pieces of information:

- Sender identity
- Recipient identity
- Message content
- Message size
- Message frequency

Attacks on the users' identity are no longer possible as the identity used on the nodes is based on eIDs instead of the users' true identity. As the eIDs may exist in parallel, overlapping, or in a serial manner and are strictly unlinked to the true identity, no statement can be made concerning the linking of ephemeral identities to the real identities. Frequency patterns or behavioral patterns may be split among multiple identities and distributed over multiple nodes.

Frequency and bandwidth analysis are not possible as the frequency and bandwidth of a single message are not trackable, and the size of a message is generally not related to the message flow. An exception to this statement is when routing a different message through a vortex system using a reused routing block general statements such as "the message is bigger than the previous one" about the size of the message is possible if the routing block makes use of relative split operations. In experiments, we were able to mimic any desired communication pattern we wanted for an adversary to be found.

The message content remains cryptographically secured if the dual trust (sender and receiver node) is not broken, the message is encrypted on the senders' node, the message is only decrypted on the node of the receiver, and remains at least wrapped in this encryption during the whole transfer.

11.1 Additional Considerations

11.1.1 Man in the Middle Attacks to Conversations

Traditional man-in-the-middle (MITM) attacks are not possible when using the *MessageVortex* protocol as the remote identity secures the recipient to a specific recipient. If, however, a recipient identity has been compromised either by stealing its private key or by injecting a wrong identity in the senders' repository man in the middle attacks become possible. We do not cover this problem within this work as a secure, verifiable way to exchange identities is not included in the protocol.

11.1.2 Identification of Participating Nodes

Participating nodes may be identified when injecting evil routing nodes. When suspecting such nodes first step should be moving outside the jurisdictional reach before reaching out to the final anonymity set. If the anonymity set is compromised, identification of the participating nodes is, however, possible.

11.1.2.1 Identification by Content

Message extraction by content is not generally possible even if knowing the blending type and corresponding blending keys. As the *VortexMessage* does not show an outer structure such as ASN.1 or similar. The message itself remains undetectable.

11.1.2.2 Identification by Query

A vortex node may be identified by the query if the node responds to unencrypted requests. An active node may be differentiated from an inactive node at any time if a valid blending specification is known. With such a specification, an evil routing node is capable of creating requests such as new identity requests. As soon as the evil node receives a reply, the node may conclude that the probed node is a VortexNode.

11.1.2.3 Identification by Traffic Type

Vortex node shows a specific behavior in terms of supported protocols. Depending on the implementation, this behavior is detectable. At the moment, supported protocols are POP/SMTP and XMPP. Since both protocols are widespread among Internet users, this footprint is shallow.

11.1.3 Storage of Messages and queues

The storage of messages sent through MessageVortex should be handled with great care. It seems, at first sight, a good idea to merge all messages in globally available storage such as the IMAP account of the receiving entity. However – In doing so, we would discover the message content to the providing party of a mail account. Since we handled the message with great care and tremendous costs up until this point, it would be careless doing so.

Storing them in a localized and receiving entity controlled storage is a good idea but leaves security considerations like a backup possibly to an end-user. This might be better but, in effect, a questionable decision. There is, however, a third option: By leaving the message unhandled on the last transport layer of the MessageVortex chain, we may safely back up the data without disclosing the message content. Merging the content then dynamically through a specialized proxy would allow the user to have a unified view on his without compromising the security.

Part IV

Discussion

In the following chapters, we analyze the protocol thoroughly for fitness of purpose.

We first apply an analysis of the protocol to identify all pieces of information leaked at all levels.

Then we apply a dynamic attack analysis of the protocol to identify all meta pieces of information leaked during transmission of the protocol such as timing or context between messages. We analyze this by assuming behaving and misbehaving nodes.

We distinguish between passive and active adversaries. Passive adversaries follow the MessageVortex protocol, have unlimited observation capabilities on the network up to layer 4 of the ISO/OSI protocol, and do have unlimited observation capabilities on the transporting layer of MessageVortex. We assume that an observing adversary, as defined in 4.1, is part of these adversaries. Active adversaries share the capabilities of passive adversaries, but do not follow the MessageVortex protocol. Both adversaries try to obtain valuable information (e.g., message content, metadata such as the communicating peers or message frequencies).

We then sum up the achieved goals by looking at well-known attacks and analyze the effectiveness of them on the protocol.

At the very end of this chapter, we identify the gaps uncovered by this work.

12 Protocol Analysis

In this section, we analyze the protocol statically. Looking at a full message, we get the protocol outline, as shown in (12.1) on page 52.

12.1 Transport and Blending Layer

12.1.1 Analysis of Plain Embedding

It is undeniable why a file treated with plain embedding is easily identifiable as a broken or tampered file. While the information remains parseable, its content is no longer sensible to a human and thus at least suspect.

We wanted to know if there is a simple method to detect the modifications of such a file. While most of the analysis method requires the processing of large data sets, we tried to find apparent, non-calculation-intense test methods that were generic. We did not take any content-based characteristics such as “outline of an image” or “resulting spectrum of a sound file” into account. As our embedding is generic, we searched for a similar detection method.

A property of encrypted ciphertext is the high entropy. We, therefore, used the calculation of the Shannon entropy in bytes as property and tried to show the shift of entropy within the files. This detection method depended very much on the type of file used for embedding. It showed an expected behavior, that file types having in the expected area a similar entropy were not detectable by this method. However, we identified some file types to be unsuitable for plain blending due to their entropy structure.

We analyzed the files by calculating the entropy of blocks 256 bytes with a sliding window over a randomly collected set of images (e.g., the first 100 entries of a file type after searching for “mouse”, “cat”, “camel”, or “dog”). We did intentionally not filter or eliminate images. Surprisingly, we were able to tell file types apart, were able to identify files with thumbnails or an interlaced structure. We even identified certain specific patterns regarding the producer type of an image (e.g., we could differentiate between pictures scanned or taken by a camera). It was not so much surprising that we were able to identify these features, but the fact that we could see them in entropy data.

We then carried out an analysis identifying the typical entropy and the inner structures. The graphs in 12.2 show a typical analysis. In that case, we looked at 100 images of each type. We graphed and analyzed their entropy and tested for the suitability of a plain embedding. Table 12.1 lists the average entropy of analyzed file types and makes remarks about the suitability for plain embedding. In practice, we found that most suitable file formats have an entropy of ≈ 7.2 and an interquartile range (IQR) of 0.15 or less. Furthermore, files should have a big, uniform, the non-structured range containing these characteristics. Such a file has a suitable space for embedding. For reference, Figure 12.3 shows the distribution of typical MessageVortex blocks. We did find that the entropy must be uniformly matched in the case of plain embedding.

When blending into images, BMP showed a strongly varying entropy within a file. A sampling of ten blocks at random position resulted already in detection with a failure rate below 5%. PNG and JPG files showed to be very robust within the sample. We did not succeed in identifying the MessageVortex blending content based on entropy values. GIF images showed to be unsuitable. Archive formats such as zip files were extremely robust. We were able to embed it into a zip file and marking it (generically) as an encrypted password file. This embedding was genuinely undetectable. However, such embedding may potentially lead to censorship based on blacklisting.

OGG and MP3 are suitable. However, we were able to detect the entropy difference when taking extreme dense samples. These formats may, however, be suitable for not yet standardized forms of steganography. While PDF has low entropy and a high IQR typically, some parts of the files are very well suited for embedding. Plain embedding with knowledge of the format was even possible without affecting the visual result of the file.

We could show that with an approach based on Shannon entropy, we may identify plain embedded MessageVortex blocks in BMP and WAV files. Most of the file formats analyzed were performing well. They required a multitude of samples to detect the presence based on the entropy property.

All movie formats were performing similarly to jpg and PNG. However, due to the very complex structure with scattered blocks, they seem to be unsuitable for plain embedding. They are, however, strong candidates for steganography.

12.1.2 Identifying a Vortex Message Endpoint

Depending on the blending method, a single, identifiable message is sufficient to identify a VortexNode. Detectability depends on various factors, such as:

- Broken internal file structure (due to plain blending)
- Uncommon high entropy in a structureless file
- Unrelated message flow (see [**oakland2013-parrot**])
- Non-human behaviour on the transport layer (e.g., message traffic 24x7)

If an endpoint is successfully identified, then all peering endpoints of the same protocol may be identified as well by following the message flow. This does, however, not enable an adversary to inject messages as the host key is not leaked.

Assuming a global observer as an adversary and unencrypted traffic, he might discover the originating routing layer and thus identify it as Vortex node by following traces of the transport layer. In most protocols, however, this address is spoofable and not a reliable source for the originating account.

12.2 Senders routing layer

A sender may have some knowledge about the Routing block size and may, therefore, guess the complexity of the routing path. He is, however, unable to gain any additional information such as time of travel or number of hops until the target.

$$
\begin{aligned}
VortexMessage =& \langle \mathbf{MP}^{K^{-1}_{hostN}}, \mathbf{CP}^{K^{-1}_{hostN}}, \mathbf{H}^{K_{senderN}}, E^{K^{-1}_{senderN}}(H(\mathbf{HEADER})) \\
& \left[\mathbf{R}^{K_{senderN}}\right], [\mathbf{PL}] * \rangle^{K_{peerN}} \rangle && (12.1)\\
\mathbf{MP}^{K^{-1}_{hostN}} =& E^{K^{-1}_{hostN}}\left(\mathbf{PREFIX}\langle K_{peerN}\rangle\right) && (12.2)\\
\mathbf{CP}^{K^{-1}_{hostN}} =& E^{K^{-1}_{hostN}}(\mathbf{CPREFIX}\langle K_{senderN}\rangle) && (12.3)\\
\mathbf{H}^{K_{senderN}} =& E^{K_{senderN}}(\mathbf{HEADER}) && (12.4)\\
\mathbf{HEADER} =& \langle K^{1}_{senderN}, serial, maxReplays, validity, [requests, requestRoutingBlock], \\
& [puzzleIdentifier, proofOfWork] \rangle && (12.5)\\
\mathbf{R}^{K_{senderN}} =& E^{K_{senderN}}(\mathbf{ROUTING}) && (12.6)\\
\mathbf{ROUTING} =& \langle [\mathbf{ROUTINGCOMBO}] *, forwardSecret, replyBlock \rangle && (12.7)\\
\mathbf{ROUTINGCOMBO} =& \langle processIntervall, K_{peerN+1}, recipient, \mathbf{nextCP}, \mathbf{nextMP}, \\
& \mathbf{nextHEADER}, \mathbf{nextROUTING}, assemblyInstructions, id \rangle && (12.8)\\
\mathbf{PL} =& \langle \text{payload octets} \rangle * && (12.9)\\
& && (12.10)
\end{aligned}
$$

Figure 12.1: Detailed representation of a VortexMessage

Type \ Criteria	Avg. Entropy	IQR	Remarks
JPG	7.008	0.097	–
PNG	7.116	0.086	–
GIF	6.978	0.194	–
BMP	2.997	4.964	not suitable
PDF	6.660	0.282	Hard to embed due to a very complex inner structure but well suited
MP3	7.076	0.091	–
WAV	4.777	0.927	–
OGG	7.104	0.093	relatively easy to embedd. Hard not to break the file structure.
mpg4	n/a	n/a	good to embedd. Steganography could be applied here easily too.
zip	7.148	0.080	easy to embedd when using “password protected” archives
MVaes	7.176	0.072	Without length padding as reference encrypted with AES 256 CBC
MVcam	7.175	0.070	Without length padding as reference encrypted with Camellia 256 CBC

Table 12.1: comparison of protocols in terms of the suitability criteria as transport layer

12.3 Intermediate node routing layer

An intermediate node does know all the operations applied and the immediate next hop. It does learn the routing addresses of the immediately following endpoints but is unable to use these endpoints. This is because he has no means to get the host key required to communicate.

If a routing block is repeated, a router may identify the routing block as repetition. Identifying the repetition of a block can be done by looking at the serial number of replay protection. We then may give a rough estimate of the message size by comparing the payload chunks. This estimate is, however, very rough as it is bounded by the block size of the symmetrically applied encryption.

12.4 Security of Protocol Blocks

To analyze the security of the protocol, we first go through all protocol blocks. After that, we will look at the possibilities of block recombinations and how to gain data or services based on such behavior.

Assuming plain embedding, the presence of a chain of blocks may leak an existing VortexMessage. At the moment, the protocol expects at the offset and the size of the bytes to be skipped to the next block. The encoding does not assume an end of the chain marker as such a marker would make the design identifiable. As an encoding scheme, a variable byte length has been chosen. This guarantees that any file will always result in a valid chain of blocks and thus not leak such a presence.

The entropy of the only two blocks in this stream (MPREFIX and InnerMessageBlock) is comparable as both blocks are encrypted. Both blocks are encrypted and feature a similar entropy. The blocks follow each other without any delimiter. This results in a continuous stream of data with constant properties.

To avoid repeating patters at the beginning of streams due to reused identity blocks, a MURB must provide sufficient peer keys and prefix blocks. A VortexNode may, however, refuse to process MURBS (only accept maxReplays equal to 0).

All blocks of InnerMessageBlock are protected by the peer key $E^{K_{peer}}$. The forward secrets in all blocks except the payload blocks make sure that the recombination of blocks does not work for an adversary. To be successful, an adversary requires to know the forward secret of the next hop.

To keep the secrets of the next hop, hidden from the host assembling the message, the subsequent header and the routing block are protected by the sender key $E^{K_{sender}}$. A message assembling node is, therefore, not even capable of creating its own messages to an unknown node as the hosts' public key $E^{K^{1}_{host}}$ is not derivable from a message.

Therefore, a routing node is not able to assemble messages for a specific host on the base of a routed message only. A routing node does not gain any additional knowledge except for the locally executed operations, the number of messages of the ephemeral identity, the size of messages of any ephemeral identity, the sending IP of a received VortexMessage and the transport endpoint address of any receiving endpoint. The most critical information is endpoint data, as all other data is unrelated to the original message (sender recipient and size). This information becomes absolutely crucial if assuming a censoring adversary. Therefore, a sender in a jurisdiction where the use of MessageVortex is deemed illegal must use only trusted nodes within the jurisdiction and at least for the first hop outside the jurisdictional reach of an adversary.

13 Dynamic Attack Analysis

In the dynamic analysis, we reach out to an active adversary. An active adversary modifies traffic in a non-protocol conformant way, or misuses available or obtained information to disrupt messages, nodes, or the system as a whole.

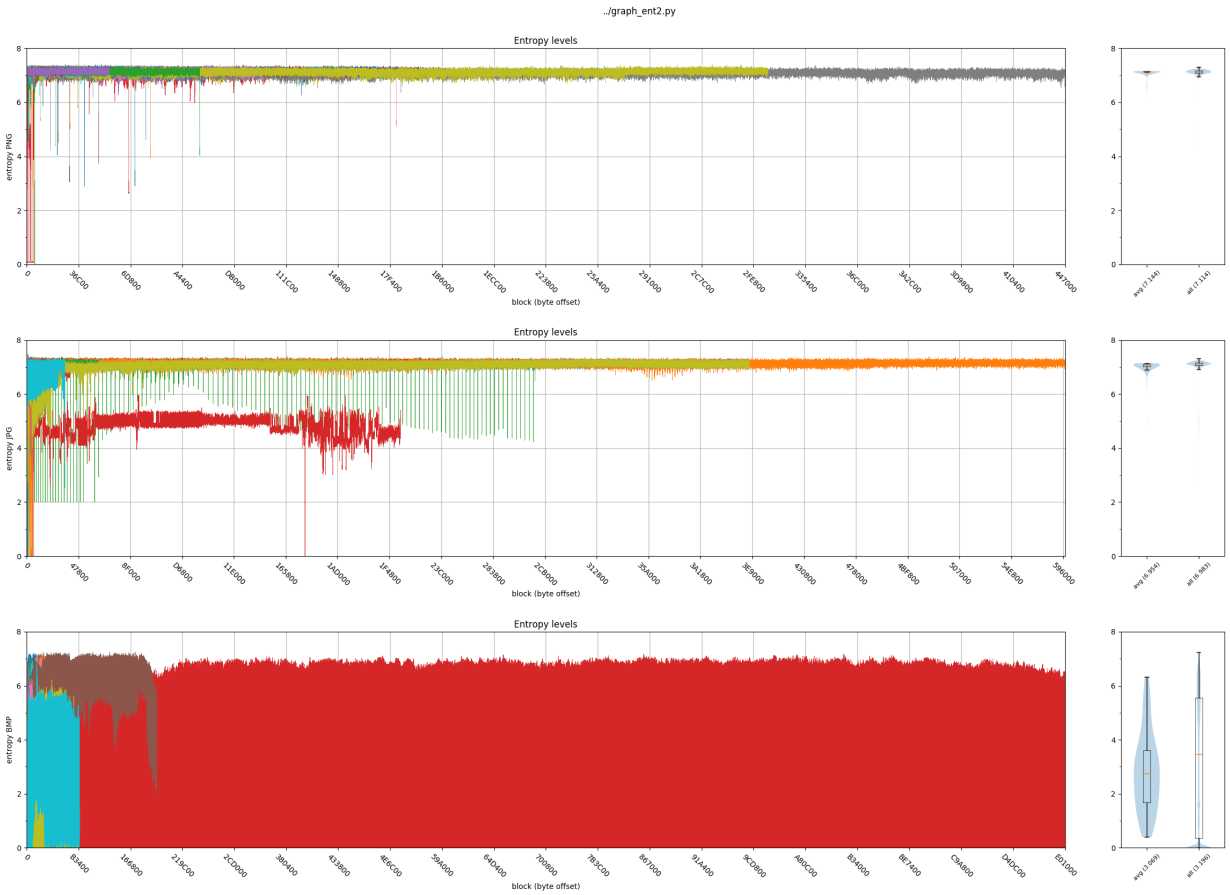

Figure 12.2: Distribution Analysis of Different, Common Graphics Formats

13.1 Attacks against the vortex system itself

An active adversary may attack the transport layer. Most of the transport layers are not able to react upon message flooding. Therefore, it is easy to attack a transport layer with a flooding attack, such as a distributed denial of service (DDoS) attack. Due to the nature of the protocol, we are unable to create additional protection on the transport layer as such modification would require a modification of the transport layer. The Vortex Message format itself is, however, crafted in such a way that only minimal effort is sufficient to get the involved parties of a transmission. The Operations $K_{msgN} = D^{K^{1}_{host}}(P)$ and $HEADER = D^{K_{msgN}}(H)$ are sufficient to identify message senders. Unknown Senders may be discarded without further processing. Known senders may be identified as legitimate and processed further. Known misbehaving identities and message duplicates may be discarded.

13.1.1 DoS Attacks against the System

An active adversary may not follow the protocol and modify any parts of the message. The following paragraphs reflect different kinds of behavior and how they affect the messages and the system as a whole.

An adversary may not follow the blending specification. If he uses a less secure specification, an independent third party observer may follow traffic. This is not sensible as such a node may send all the knowledge to such a collaborating node directly. In the case of a target node not supporting the chosen blending method, the partial message path becomes interrupted. A possible redundancy in the path may recover the message from such a case.

13.1.1.1 DoS by Traffic Replay

Traffic replay is a common way to highlight traffic in many systems by replaying the same traffic and increase the signal to noise ratio of a system. In our case, we can use the replay of a VortexMessage block to increase the traffic to a node. After decoding the header, a MessageVortex node identifies the block as a repeated block and rejects further processing.

An adversary may replay blocks with varying content. This will not result in a DoS attack as the quota is not decreased on replayed messages (see Figure 9.2).

13.1.1.2 DoS by Traffic generation

An adversary may first collect identities and quotas and use them later in a coordinated attack to force the node processing. The adversary may increase the impact by using large payloads and processing them in a costly manner. A possibility is to make extensive use of $addRedundancy$ or encryption operations. Furthermore, an attacker may attack the memory by distributing the message throughout the workspace to exhaust the routers' runtime memory.

As a router is free to process the operations of identity, he may discard an ephemeral identity and all associated resources at any time. Misbehaving or suspected misbehaving nodes may, therefore, be stopped. On the other hand, we are unable to prevent an adversary from allocating new identities. We may, however, work with multiple local host keys and distribute them according to the trust. A known party or someone trusted by them might receive a key different from a publicly advertised key. This identity key may be dropped at any time and distributed to further parties again with an identity update. We may even subdivide trusted parties into several groups by updating them with different new host keys to identify misbehaving routers without knowing them.

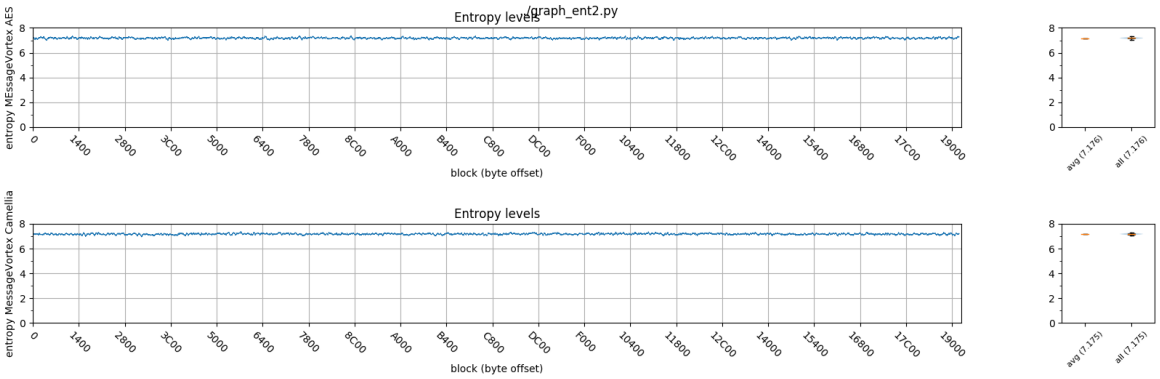

Figure 12.3: Distribution Analysis of a MessageVortex Block

13.1.2 Attacking a single ephemeral Identity of a MessageVortex Node

13.1.2.1 Denial of Service by Exhausting Quotas or Limits

A malicious node may try to exhaust quotas or limits. As we do trust in the sender and recipient, all other nodes do not know the forward secrets used in the message. The options for an adversary are then as follows:

- Resend a MURB (with different content) as often as possible to exhaust message and transfer quota.
- Create intentionally huge, incorrect message content to exhaust transfer quota.

13.1.3 Attacking Sending and Receiving Identities of the MessageVortex System

The most valuable goal of an adversary is breaking an entity's anonymity or monitor their traffic by the content or the metadata. In the following sections, we analyze the possibility of

13.1.3.1 Traffic Highlighting

Traffic caused by a routing block may be observed to a certain extent on a statistical base. A node may generate bad message content of exceptionally large or small nature. This might potentially highlight messages involved in message routing using no split or relative split operations as well as addRedundancy operations.

13.1.4 Recovery of Previously Carried Out Operations

It is crucial that an adversary is unable to recover parameters of a previously carried out operation. We analyzed though the protocol operations carefully to be sure not to leak any of the parameters. Some operations leak apparent data such as an encryption operation with a block cipher does typically leaks its block size. This has, however, been classified as invaluable data as the block size does not result in any information gain usable for attacking the system or narrowing down efforts. In figure **??**, we can show that the parameters are visible. We took the same 10kb block and treated it with all possible combinations of operation parameters. The image shows that there is a possibility of guessing the parameter with a high probability. For guessing the average Monte Carlo Pi and the average Shanon entropy in bits per byte were already sufficient. The results got a bit less clear when applying the same operation to random blocks while doing the analysis.

We have, however, found a flaw in the$addRedundancy$ operation. When applying this operation to an encrypted block, the entropy of the resulting block leaks some of the parameters of the operation. As a result of this finding, we added a custom padding and an additional encryption step. The repeated analysis showed that the operation does no longer leak these parameters through this channel.

13.2 Achieved Anonymity and Flaws

13.2.1 Measuring Anonymity

It is tough to measure anonymity, as it involves many uncontrollable factors. We may, however, control the degree of anonymity according to the number of involved parties. Assuming a sender knows the complete message path, including all operations carried out on any untrusted node a message travels through, the anonymity is maxed to the number of involved nodes n, excluding the sender nodes. This degree of $n-1$ may be further reduced if all well-known "routing only" or at least "routing mostly" nodes are reduced. Under these harsh assumptions, the set may be reduced to the potential set of "well known" recipients of a message.

We have to differentiate between several problems. An adversary has to identify the participants of an anonymity system. Then he has to identify members of a message or a communication anonymity set. Starting from there, he has to identify message flows and detect senders and receivers of messages within an anonymity set (which is not doable in all cases). If any adversary achieves this, we have to consider the anonymity to be broken. Depending on the degree of anonymity required, which is influenced by external factors, the participation in any or a small enough set may be sufficient to suffer consequences.

13.2.2 Attacking Routing Participants

While very hard in our case as we do not have "dedicated" anonymization infrastructure, It might be possible to identify members of the routing network. This due to flaws in the blending layer. While it is possible to scare off or block members of a routing network. It is far harder in a network where the members are mobile. Any user may change at any time the identity, including the endpoint, without losing its known peers. This unique property makes the participating entities very mobile and allows them to switch servers at any time without losing contact with peers for subsequent communication.

Routing participants may be identified either by publicly available information (e.g., published routing address) or by identifying unique properties of the protocol. Transport layer provider may then be forced to deanonymize the customer related to the account (if possible), or the relating account on the transport layer may be blocked.

To counter a possible threatening deanonymization, a MessageVortex node owner must maintain anonymity towards the transport layer provider. Nowadays, this is easily done in the XMPP protocol.

The account is typically not linked to any subsequent user information, such as telephone or email. Email accounts are more restrictively regulated. Providers providing accounts without registration of phone numbers or subsequent email addresses do exist (e.g., Yandex) but are rare. In both cases, a user might be identified by its IP address. This is why concealing its IP address while connecting to the transport layer is an advisable practice. Using Tor when accessing the transport layer may suffice to do so. The anonymizing service has to be strong enough to conceal the IP. The protection of the traffic itself is not required as it is already protected.

13.2.3 Attacking Anonymity through Traffic Analysis

As traffic and decoy traffic and decoy traffic are chosen by the creator of the routing block, frequency patterns cannot be detected, unlike the router did create them. The same applies to message sizes and traffic hotspots. When reusing the same routing block, eventually message sizes or general estimates such as “bigger” or “smaller size” can be made.

For an evil routing node, even paired with a global observer, it is hard to extract any useful information. An adversary might identify all messages following through it as messages of the same true identity. As ephemeral identities are short term identities, this is of limited values. By monitoring the endpoints used by an ephemeral identity, we might calculate a “likelihood of matching” for two ephemeral identities. Luckily this is not doable without allowing a high factor of uncertainty. This matching does not improve when combining multiple ephemeral identities over time. The matching might slightly improve when trying to match ephemeral identities on different routing nodes. Making strong statements about those likelihoods is not possible as we did intentionally not define a specific behavior. We may safely say that the possibility of deanonymization is degrading if using short-lived ephemeral identities.

The knowledge a node may gain from ephemeral identities is minimal. The ephemeral identity is created by a node unknown to the receiver of the request. The only thing we know is what node was adjacent when creating the ephemeral identity. As the creation of an ephemeral identity is not linked to any other identity or ephemeral identity relationship between ephemeral identities on two nodes cannot be established. If two adjacent nodes cooperate when processing two linked ephemeral identities, no additional knowledge may be won. If two collaborating nodes have one or more non-collaborating nodes between them, they lose all linking knowledge due to the non-collaborating nodes.

Operations have been carefully crafted to leak as little information as possible. Being able to encrypt or decrypt a payload block does not leak any information. The data processed may be true message traffic or decoy as we do not know what the nature of the received message was. If an RBB avoids repeating patterns of blocks on nodes, it is not possible to link ephemeral identities of two non-adjacent nodes. Repeating patterns may arise, for example, if a block pb_1 is decrypted and re-encrypted on two nodes. In this case, both nodes may match the message as it contains the same content between the operations.

node f:

$$pb_2 = D(pb_1$$

$$pb_3 = E^{K_t}(pb_2)$$

node f+1:

.

.

node f+x:

$$pb_4 = D^{K_t}(pb_3)$$

In this example the patterns of $pb3$ and $pb_4 = pb_2$ are two patterns repeating on non-adjacent nodes. The same conclusions are even more valid for splitting operations. These two operations should be regarded as helpers for the *addRedundancy* and *removeRedundancy* operations. These operations may be used to generate decoy traffic or to destroy data without knowledge of doing so of the processing node. If we process a function $addRedundancy_{2of3}$, any of the output blocks contains the input payload, and any two of them may be used to recover the data. At the same time, an operation $removeRedundancy_{2of3}$ may be successful or not. The node is unable to differentiate between the two states. The padding applied and the unpadded encryption makes it impossible to judge upon success or fail of an operation.

As the communication pattern is defined by the RBB and not always the same, it is hard to judge on the security. We may, however, look at some generic examples and show that we can achieve the goals byzantine fault tolerance, privacy and unlinkability, and anonymity. Figure 13.1 shows a sending node s, a series of routing nodes n_i, j assembled to routing chains. Furthermore, we have a r for which the message is destined and a set of nodes a_k building the anonymity set. Neither the number of chains j nor the length of the chains i is relevant. A node or a sequence of nodes may be part of multiple chains. By normalizing a path into such a form, we may at least analyze some properties of the protocol. We furthermore have to keep in mind that we trust sender s and receiver r. Any possible routing block may be reduced to this scheme if knowing the exact building instructions applied by the RBB.

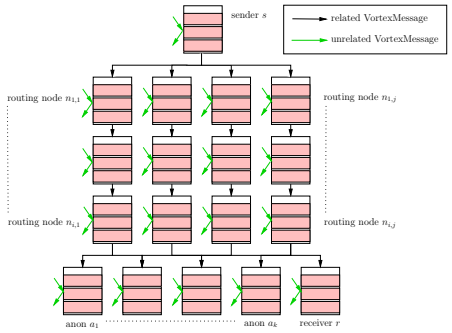

Figure 13.1: A possible path of a VortexMessage

We have to consider the fact that two adjacent nodes collaborating may build one combined workspace executing all operations. They are, therefore, able to link all operations of these two adjacent nodes and follow all incoming and outgoing paths. We, therefore, may assume that two adjacent nodes or an uninterrupted series of collaborating nodes may be substituted by one node.

So a routing node $n_{1,}$ may not know if a *VortexMessage* received from s is the result of processing another message or the message has been injected on node s. Furthermore, if s was acting as a routing node, it successfully unlinked the message from any previous node. The sending node s may send a message by first employing an *addRedundancy* operation or splitting and encrypting the message. Each path through the streams has then not enough information to rebuild the combined message. If employing an *addRedundancy* operation, a receiver r may recover a message, if sufficient paths through the routing nodes were acting according to the protocol. Paths with misbehaving nodes may eventually be identified depending on the number of redundancy operations. Assuming that the RBB included proper padding Information for the receiver r, the receiver may identify what set of *VortexMessages* leads to the original message due to the padding applied before the RS function. So if sufficient paths, depending on the chosen operations at r, provide correct data, we may recover nodes misbehaving in our paths. If one node in a path is not collaborating with adjacent nodes in the path, the path of the *VortexMessage* becomes unlinked as previously shown with sender s. If multiple paths are used, all paths must have at least one honest node to unlink the message.

If all nodes in the anonymization set $a_1 \ldots a_k$ are honest, any preceding node may not know whether the message ends at that node or the message is just routed through an honest node. Even if some of the anonymization nodes are not honest or collaborating with an adversary, the anonymity set may be reduced in size, but the receiver

is still part of the anonymity set spanning the honest anonymization nodes. So, we have shown that depending on the chosen routing block, anonymity, unlinkability, and fault tolerance against a misbehaving node may be achieved. AN RBB may furthermore send additional *VortexMessages* to suspected misbehaving nodes. If misbehavior is reproducible within an ephemeral identity, the RBB may identify it by picking up parts of the previously sent message and comparing them to an expected state. An RBB may even introduce message paths leading back to the RBB itself. Such a message path may allow observation of the progress and success of the message delivery.

13.2.4 Attacking Anonymity through Timing Analysis

Timing is under full control of the routing block builder. No information can be derived from the timing. This is even the case if a routing block is reused. The precise timing on the network depends additionally on other factors, such as delaying through anti-UBE or anti-malware measures or delays through local delivery between multiple nodes.

13.2.5 Attacking Anonymity through Throughput Analysis

Increasing the throughput to highlight a message channel is not possible since the replay protection will block such requests. It may be possible for a limited number of times by replaying a MURB. This is one of the reasons why the usage of MURBs is discouraged unless necessary.

13.2.6 Attacking Anonymity through Routing Block Analysis

The routing block is cryptographically secure. The size of the routing block may leak an estimate about its inner complexity. It does not reveal any critical pieces of information like remaining hops to the message end or target or similar.

13.2.7 Attacking Anonymity through Header Analysis

The header contains valuable data that is cryptographically secured and only visible to the next receiver.

To an adversary not knowing the key, the size of the prefix block may leak the key size. The size of the header block itself may leak the presence of any optional blocks. Besides that, no other information is leaked to such an adversary.

To an adversary knowing the decryption key (evil routing node), the content of the header block is visible. This header block leaks all routing information for the respective node and thus the ephemeral identity. This block leaks some information of minimal value. It may leak the activity of an ephemeral identity, including frequency. This activity is, however, only matching the minimal activity of an endpoint identity as an endpoint may have multiple ephemeral identities on one node.

13.2.8 Attacking Anonymity through Payload Analysis

The payload itself does not leak any information about the message content. All content is cryptographically secured. Content may, however, leak the block size of the applied cipher.

13.2.9 Attacking Anonymity through Bugging

Bugging is one of the most pressing problems. The protocol has been carefully crafted not to allow any bugging. The use of MIME messages in the final message, however, enables the bugging of the message itself. A bugged message content may breach receiver anonymity to the sender of the message.

13.2.10 Attacking Anonymity through Replay Analysis

Due to the replay protection, no traffic may be generated or multiplied except for the traffic sent by the attacking node. As this information is already known to the node, there is no value in doing so.

13.2.11 Diagnosability of traffic

13.2.11.1 Hijacking of Header and Routing Blocks

An attacker might try to recombine a header block of the third party with a routing block crafted to get the workspace content of a different node. To protect against this scenario, every routing block and its corresponding header block has a shared value called forwardSecret. As the content of a hijacked header block is not known, he is unable to guess the forward secret within the block.

It is not possible to brute-force the value due to the replay protection. More precisely, the probability of hijacking a single identity block is $\frac{1}{2^{32}}$. Hijacking such a block allows onetime access to the working space and is visible to the owner due to the manipulated quotas. Failing an attack will result in deleting the ephemeral identity, and a new, unlinked ephemeral identity will be created.

13.2.11.2 Partial Implicit Routing Diagnosis

We can create data that is routed back to or through the original sending node. This traffic is well defined and may be used to certify that the loop processing the message is working as expected. By combining the messages and sending intermediate results through multiple paths, it is even possible to extract the sub status of some loops and combine the result within transfer into a single message.

As a special case, a sender may use implicit routing diagnostic to diagnose the full route. A sender may do this by taking specific excerpts of the received message at the recipients' node and route these blocks back from the recipient to the sender.

13.2.11.3 Partial Explicit Routing Diagnosis

If a message fails to deliver according to implicitly routing diagnosis, additional messages may be sent to pick up the content of the workspace of ephemeral identities throughout the path. These messages are due to the only binding to the ephemeral identity, not distinguishable from the original messages. Assuming that a node always behaves either according to or not according to the rules of the system, a node may be identified by capturing built blocks with known content.

If a node is identified as a misbehaving node, it may be excluded from subsequent routing requests or reduced in its reliability or trustability ratings. A node may calculate such scores locally to build a more reliable network over time, avoiding misbehaving or non-conformant nodes. This does not violate our zero-trust philosophy as the scoring is made locally and relies on our observations.

14 Recommendations on Using the Vortex Protocol

The following sections list recommendations using the VortexProtocol. It is a summary of the previous sections.

14.1 Reuse of Routing blocks

Routing blocks should not be reused. The reuse of a routing block may leak some limited information to an adversary node such as approximate message size or message frequency of an unknown tupel using this network.

14.2 Use of Ephemeral Identities

Ephemeral identities should be used for a minimal number of messages. Using multiple identities with overlapping lifespans is considered a good practice. Using different ephemeral identities for the same message is acceptable and can be a good practice as long as operations do not leak the linking between those two identities.

Special care must be taken if using overlapping ephemeral identities across nodes. While ephemeral identities may be completely unlinked on a single node, the linking between multiple nodes may leave a trace from one identity to the next. It is advisable to recreate on a regular base all ephemeral identities from scratch. This guarantees an unlinking from previous ephemeral identities.

14.3 Recommendations on Operations applied on Nodes

All operations, carried out on a single node, have to be crafted in such a way that no information whether the operation is a decoy or a real message is leaked. Otherwise, it becomes possible to narrow down the message flow.

Encryption operations should be either strictly encrypting or strictly decrypting. At no point in the path, a previously applied encryption on an untrusted node should be removed as removal might lead to linking to the previous inverse operation.

Similarly, there are rules for adding and removing redundancy information. As these operations serve as decoy traffic generators, great care needs to be taken not to leak this information. We emphasize here again that it is possible to add redundancy information on one node, encrypt one or multiple blocks once, or multiple blocks on a second node, and then remove the redundancy information again from the new set. This will lead to a payload data block than the original. However, this does not qualify the block as decoy traffic. The process may be reversed on the final recipient. Such an operation is, however, mathematically very demanding if the same operation is used for redundancy at the same time as multiple possible tuples need to be tried if one node has failed.

Whenever possible, the reappearance of a payload block in a single encoding it should be avoided or limited to an absolute minimum as such an occurrence allows linking of two ephemeral identities.

14.4 Reuse of Keys, IVs or Routing patterns

An RBB should avoid reuse of any keys, IVs, routing patterns, or PRNG seeds along its routing path of untrusted nodes. Reusing such values would allow an attacker to match ephemeral identities to a single identity. While this is minimal risk and may be ignored in some cases, an RBB should avoid it as it may leak information to collaborating nodes.

14.5 Recommendations on Choosing involved Nodes

Involved nodes should be trustworthy but not necessarily trusted. A message should always include a set of known recipients. It is regarded as a good practice to use a minimal fixed anonymity set of known recipients as routers. Doing so does not leak any information unless always the same pattern of operations is applied (see 14.1).

14.6 Message content

Although it is possible to embed any content into a Vortex message, great care should be taken as the content may allow disclosing a reader's identity or location. For this reason, only self-contained messages should be used (such as plain text messages).

Allowing a user to use more complex representations such as MIME offers many possibilities for the bugging of the content. A client displaying such messages should always handle them with great care. Taping messages by downloading external images or verifying the validity by OCSP or even doing a reverse lookup on an IP address may leak valuable information.

14.6.1 Splitting of message content

Message content should be split and distributed among routing nodes. Splitting should, however, not be done excessively to avoid failure due to too many failing nodes. It furthermore makes diagnostics complicated.

14.7 Routing

14.7.1 Redundancy

Redundancy is a valuable feature of the protocol. It allows unsuspicious decoy generation and to compensate message path disruption. A routing block should always be crafted in such a way that redundancy is aligned with the complexity of the routing block and the importance of a message to avoid an adversary controlling all nodes except for the sender's and receiver's one.

14.7.2 Operation Considerations

Operations should be kept easy, but at the same time, guarantee anonymity. The following recommendations are kept to an absolute minimum in order not to create any identifiable behavior.

A payload block should always have a single representation only once when traveling through routing nodes. A recurring pattern would allow an evil router to identify and thus match an ephemeral identity of one router to an ephemeral identity of another router even if there are multiple routes in between. So, when applying encryption only operations between routing nodes, the encryption should be onionized. A clear onionizing routing pattern (only showing encryption steps on a single chunk) is OK. A pattern such as removing encryption and then reapply different encryption is not.

14.7.3 Anonymity

Anonymity is greatly dependent on the quality of the routing block and the chosen anonymity set for a single message and a communication tuple over time.

14.7.3.1 Size of the Anonymity Set

The requirement for an anonymity set is dependent on jurisdictional restrictions. In some of the more restrictive countries, no one can be held guilty for an action that may not be credibly assigned to him alone. In other jurisdictions, it is possible to be held liable for actions just because of an identified membership in a group. This makes it essential that message traffic and the crafting of the blending is under the sole control of the sender. He needs to create an anonymity-set sufficiently large and spanning enough jurisdictions to create sufficient anonymity for his situation.

15 Missing gaps to be covered in future analysis

The current blending layer is simple in its inner working. It creates context-less messages based on an easily recognizable scheme. An unsuspecting observer may have the impression that this is just a way of communicating, but censor may, by observing the message flow easily and conclude that these messages are not written by a human. Such detection could lead to censorship of the respective routing node and thus disrupt the message flow. It is easy to recover from such censorship by advertising a new identity to known peer partners. To minimize the effects of censorship, an improvement in this area would help.

To be undetectable, all work done by the blending layer has to be indistinguishable from regular human communication. This applies not only to the message steganographic embedding of the message but to the message content as well. This is very much similar to the problems of chatterbots these days. Assuming that a blending layer is only communicating with other nodes correctly embedding messages, we have a chatterbot problem. It is reduced as the chatterbot must only reply credibly and undetectable to generated messages of other chatterbots. If assuming that a blending layer replies to any non-Vortex nodes, the problem boils down to a Turing test, as stated in [**turing1950computing**]. As we defined that an adversary has enormous but limited resources, this blending is, however, sufficient if it is done "good enough". What criteria would apply here is a topic for further research. Applying any research to this topic would require to add a more precise adversary model.

The currently applied choice of transport layer protocol is a snapshot of current Internet traffic. While done with great care, it must be adapted to the changing communication habits of humanity. Identifying new or depreciated communication protocols and blending schemes would be another field of research.

A comprehensive survey of the newest trends and techniques in steganography is another topic to be covered. It would allow identifying new candidates for blending techniques. Especially interesting are steganography algorithms covering movie file formats.

This is especially hard since true evidence of in-depth protocol usage seems to be completely missing. While we were able to gather much data which is collected by simple routers (such as bandwidth), credible figures about client and content usage seemed to be completely missing or of very poor quality.

Anonymity has effects on the behavior of humans. We have found that although there is some research in this field (such as [**postmes2001social**]), the evidence is very weak. Although the possibility of anonymity is undisputed among so-called free countries, the downsides (e.g., misuse for criminal acts) of anonymity are apparent. More research in this field is required. On the other hand, a lack of awareness for anonymity, especially in "non-free" jurisdiction, has been observed, which would be another relevant field of research.

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.

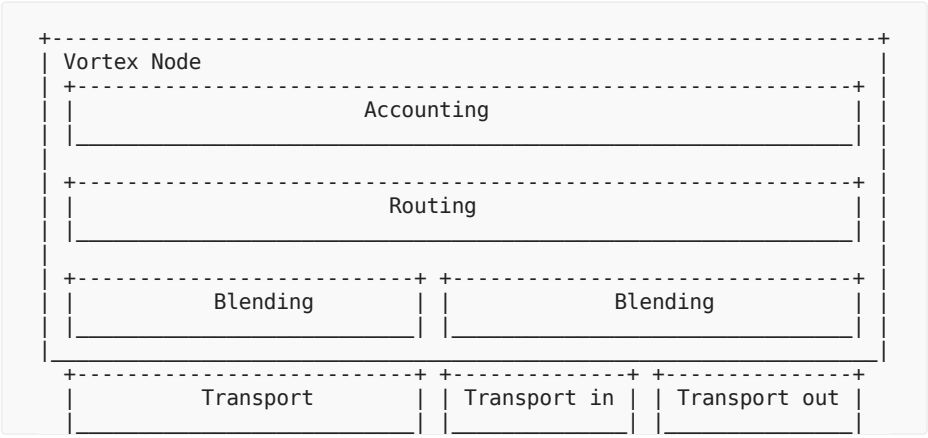

Figure 1: Layer overview

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.

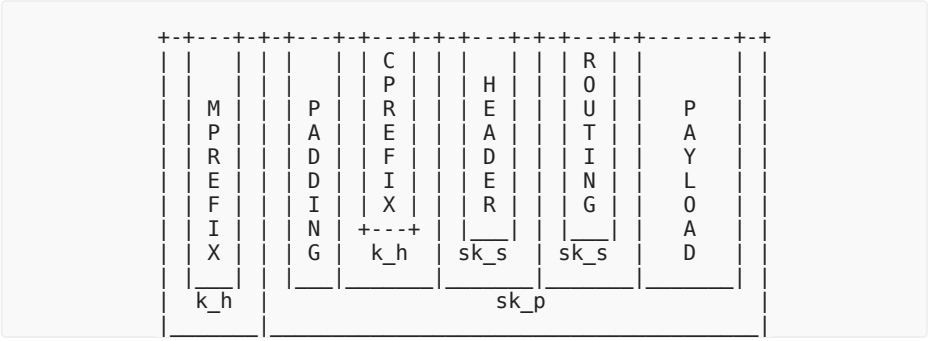

Figure 2: Vortex message overview

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

B Analysis on Common Internet Protocols Suitable as Transport Layers for MessageVortex

B.1 Introduction

The following sections list common Internet protocols. We analyze those protocols for the fitness as transport layer of message vortex.

B.2 Methods

All sections are structured the same way. We first refer to the protocol or standard and describe it in the simplest possible form. We refer to subsequent standards if required to consider extensions where sensible. We then apply the previously referenced criteria and make a concise summary of the suiting of the protocol as a transport layer. The findings of this section is listed in table 5.1. The list here does not reflect the quality or maturity of the protocols. It is a simple analysis of suiting as a transport layer.

All sections are structured the same way.

- Description
  We first refer to the protocol or standard and describe it in the simplest possible form. We refer to subsequent standards if required to consider extensions where sensible.
- Apply criteria

  We then apply the previously referenced criteria and make a concise summary of the suiting of the protocol as a transport layer. The findings of this section is listed in table 5.1. The list here does not reflect the quality or maturity of the protocols. It is a simple analysis of suiting as a transport layer.

B.2.1 Applied Criteria

- Widely adopted (Ct1)
  The more widely adopted and used a protocol is, the harder it is due to the sheer mass for an adversary to monitor , filter, or block the protocol. This is important for censorship resistance of the protocol.
- Reliable (Ct2)
  Message transport between peers should be reliable. As messages may arrive anytime from everywhere, we do not have means to synchronize the peer partners on a higher level without investing a considerable effort. Furthermore, the availability of information when what type of information should be available at a specific point in the system would drastically simplify the identification of peers. To avoid synchronization, we do look for inherently reliable protocols.
- Symmetrical built (Ct3)
  The transport layer should rely on a peer to peer base. All servers implement a generic routing that requires no prior knowledge of all possible targets. This criterion neglects centralized infrastructures. This criterion may be dropped, assuming that the blending layer or a specialized transport overlay is responsible for routing.

B.2.2 Analyzed Protocols

We were unable to find a comprehensive list of protocols being used within the Internet and their bandwidth consumption. A weak reference is [**zhou2011examining**]. This wekaness is founded in the fact that traffic in this report is callified among two critera: Know server or known port. As streaming services consume a considerable part of the Internet bandwidth (acording to the report more than 60% download). The focus on the report lies on the bandwidth intense figures. However, leavin aside all sources which are strictly one way or dominated by a small number of companies worldwide, the “top 10” list of the report shrinks to the two categories “File sharing” (Rank 5; 4.2% download and 30.2% upload) and “Messaging” (Rank 8; 1.6% download and 8.3% upload bandwidth).

In lack of such material we first collected a list of all common Internet messaging protocols (synchronous and asynchronous). We added furthermore some of the most common transfer protocols such as HTTP and FTP and analyzed this list.

- Messaging Protocols
  - SMTP
  - CoAP
  - MQTT
  - AMQP
  - XMPP
  - WAMP
  - SMS
  - MMS
- Other Protocols
  - FTP, SFTP, and FTPS
  - TFTP
  - HTTP

The following protocols have been discarded as we have considered them as outdated:

- MTP[**RFC780**] (obsoleted by SMTP)
- NNTP[**RFC3977**] (outdated and has only a small usage acording to [**kim2010today**])

We furthermore discarded all RPC-related protocols as they would by definition violate Ct3.

B.3 Analysis

B.3.1 HTTP

The HTTP protocol allows message transfer from and to a server and is specified in RFC2616 [**rfc2616**]. It is not suitable as a communication protocol for messages due to the lack of notifications. There are some extensions that would allow such communications (such as WebDAV). Still, in general, even those are not suitable as they require a continuous connection to the server to get notifications. Having a “rollup” of notifications when connecting is not there by default but could be implemented on top of it. HTTP servers listen on standard ports 80 or 443 for incoming connects. The port 443 is equivalent to the port 80 except for the fact that it has a wrapping encryption layer (usually TLS). The incoming connects (requests) must offer a header part and may contain a body part that would be suitable for transferring messages to the server. The reply to this request is transferred over the same TCP connection containing the same two sections.

HTTP0.9-HTTP/1.1 are clear text protocols which are human-readable (except for the data part which might contain binary data). The HTTP/2[**rfc7540**] protocol is using the same ports and default behavior. Unlike HTTP/0.9-HTTP/1.1, it is not a clear text but encodes headers and bodies in binary form.

To be a valid candidate as storage, unauthenticated WebDAV support, as specified in [**rfc4918**], must be assumed.

The protocol does satisfy the first two main criteria (Ct1: Widely Adopted and Ct2: Reliable). The main disadvantage in terms of a message transport protocol is that this protocol is not symmetrically. A server is always just “serving requests” and not sending information actively to peers. This Request-Reply violates criteria

(Ct3: Symmetrically built) and makes the protocol not a primary choice for message transport.

It is possible to add such behavior to the blending layer using HTTP servers as pure storage. Such a behavior would however be most likely detectable and thus no longer be censorship resistant.

B.3.2 FTP

FTP is defined in RFC959[**RFC959**]. This Protocol is intended for authenticated file transfer only. There is an account available for general access ("anonymous"). This account does normally not offer upload rights for security reasons. It is possible to use FTP as a message transfer endpoint. The configuration would work as follows: the user "anonymous" has upload rights only. It is unable to download or list a directory. A node may upload a message with a random name. In case a collision arises, the node retries with another random name. The blending layer picks messages up using an authenticated user. This workaround has multiple downsides. At first, handling FTP that way is very uncommon and usually requires an own dedicated infrastructure. Such beavior would make the protocol again possibly detectable. Secondly, passwords are always sent in the clear within FTP. Encryption as a wrapping layer (FTPS) is not common, and SFTP (actually a subsystem of SSH) has nothing in common with FTP except for the fact that it may transfer files as well.

Furthermore, FTP may be problematic when used in active mode for firewalls. All these problems make FTP not very suitable as a transport layer protocol. FTPS and SFTP feature similar weaknesses as the FTP version in terms of detectability of non-standard behavior.

Like in HTTP, a disadvantage of FTP in terms of a message transport protocol is that this protocol is not symmetrically. A server is always just "serving requests" and not sending information actively to peers. This Request-Reply violates criteria (Ct3: Symmetrically built) and makes the protocol not a primary choice for message transport. The Protocol, however, satisfies the first two criteria (Ct1: Widely Adopted and Ct2: Reliable).

B.3.3 TFTP

TFTP has, despite its naming similarities to FTP, very little in common with it. TFTP is a UDP based file transfer protocol without any authentication scheme. The possibility of unauthenticated message access makes it not suitable as a transport layer. The protocol is due to the use of UDP in a meshed network with redundant routes. Since the Internet has a lot of these redundant routes, this neglects the use of this protocol.

TFTP is rarely ever used on the Internet, as its UDP based nature is not suitable for a network with redundant routes. Not being common on the Internet violates criterion one (Ct1: Widely Adopted). TFTP is not symmetrically. This means that a server is always just "serving requests" and not sending information actively to peers. This Request-Reply violates criteria (Ct3: Symmetrically built) and makes the protocol not a primary choice for message transport. The Protocol furthermore violates Ct2 (Ct2: Reliable) as it is based on UDP without any additional error correction.

B.3.4 MQTT

MQTT is an ISO standard (ISO/IEC PRF 20922:2016) and was formerly called MQ Telemetry Transport. The current standard as the time of writing this document was 3.1.1 [**mqtt**].

The protocol runs by default on the two ports 1883 and 8883 and can be encrypted with TLS. MQTT is a publish/subscribe based message-passing protocol that is mainly targeted to m2m communication. This Protocol requires the receiving party to be subscribed to a central infrastructure in order to be able to receive messages. This makes it very hard to be used in a system without centralistic infrastructure and having no static routes between senders and recipients.

The protocol does satisfy the second criterion (Ct2: Reliable). It is in the area of end-user (i.e., Internet) not widely adopted, thus violating Criteria 1 (Ct1: Widely Adopted). In terms of decentralization design, the protocol fails as well (Ct3: Symmetrically built).

B.3.5 Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol (AMQP) was initially initiated by numerous exponents based mainly on finance-related industries. The AMQP-Protocol is either used for communication between two message brokers, or between a message broker and a client[**amqp**].

It is designed to be interoperable, stable, reliable, and safe. It supports either SASL or TLS secured communication. The use of such a tunnel is controlled by the immediate sender of a message. In its current version 1.0, it does, however, not support a dynamic routing between brokers[**amqp**].

Due to the lack of a generic routing capability, this protocol is therefore not suitable for message transport in a generic, global environment.

The protocol satisfies partially the first criterion (Ct1: Widely Adopted) and fully meets the second criterion (Ct2: Reliable). However, the third criterion is violated due to the lack of routing capabilities between message brokers (Ct3: Symmetrically built).

B.3.6 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is a communication Protocol which is primarily destined to m2m communication. It is defined in RFC7252[**RFC7252**]. It is defined as a lightweight replacement for HTTP in IoT devices and is based on UDP.

The protocol does partially satisfy the first criteria (Ct1: Widely Adopted). The second criterion (Ct2: Reliable) is only partially fulfilled as it is based on UDP and does only add limited session control on its own.

The main disadvantage in terms of a message transport protocol is that this protocol is not (like HTTP) symmetrically. This means that a server is always just "serving requests" and not sending information actively to peers. This Request-Reply violates criteria (Ct3: Symmetrically built) and makes the protocol not a primary choice for message transport.

B.3.7 Web Application Messaging Protocol (WAMP)

WAMP is a web-sockets based protocol destined to enable M2M communication. Like MQTT, it is publish respectively subscribe oriented. Unlike MQTT, it allows remote procedure calls (RPC).

The WAMP protocol is not widely adopted (Ct1: Widely Adopted), but it is reliable on a per-node base (Ct2: Reliable). Due to its RPC based capability, unlike MQTT, a routing like capability could be implemented. Symmetrical protocol behavior is therefore not available but could be built in relatively easy.

B.3.8 XMPP (jabber)

XMPP (originally named Jabber) is a synchronous message protocol used in the Internet. It is specified in the documents RFC6120[**RFC6120**], RFC6121[**RFC6120**], RFC3922[**RFC3922**], and RFC3923[**RFC3923**]. The protocol is a very advanced chat protocol featuring numeros levels of security including end-to-end signing and object encryption[**RFC3923**]. There is also a stream initiation extension for transferring files between endpoints [**xep0096**].

It has generic routing capabilities spanning between known and unknown servers. The protocol offers a message retrieval mechanism for offline messages similarily to POP [**xep0013**].

The protocol itself seems to be a strong candidate as a transport layer as it is being used actively on the Internet.

B.3.9 SMTP

The SMTP protocol is currently specified in [**RFC5321**]. It specifies a method to deliver reliably asynchronous mail objects through a specific transport medium (most of the time, the Internet). The document splits a mail object into a mail envelope and its content. The envelope contains the routing information, which is the sender (one) and the recipient (one or more) in 7-Bit ASCII. The envelope may additionally contain optional protocol extension material.

The content should be in 7-Bit-ASCII (8-Bit ASCII may be requested, but this feature is not widely adopted). It is split into two parts. These parts are the header (which does contain meta-information about the message such as subject, reply address, or a comprehensive list of all recipients), and the body which includes the message itself. All lines of the content must be terminated with a CRLF and must not be longer than 998 characters, excluding CRLF.

The header consists of a collection of header fields. Each of them is built by a header name, a colon, and the data. The exact outline of the header is specified in [**RFC5322**] and is separated with a blank line from the body.

[**RFC5321**] furthermore introduces a simplistic model for SMTP message-based communication. A more comprehensive model is presented in section SMTP and Related Client Protocols as the proposed model is not sufficient for a detailed end-to-end analysis.

Traditionally the message itself is mime encoded. The MIME messages are mainly specified in [**RFC2045**] and [**RFC2046**]. MIME allows to send messages in multiple representations (alternates), and attach additional information (such as possibly inlined images or attached documents).

SMTP is one of the most common messaging protocols on the Internet (Ct1: Widely Adopted), and it would be devastating for the business of a country if, for censoring reasons, this protocol would be cut off. The protocol is furthermore very reliable as it has built-in support for redundancy and a thorough message design making it relatively easy to diagnose problems (Ct2: Reliable). All SMTP servers usually are capable of routing and receiving messages. Messages going over several servers are common (Ct3: Symmetrically built), so the third criterion may be considered as fulfilled as well.

SMTP is considered a strong candidate as a transport layer.

B.3.10 SMS and MMS

SMS capability was introduced in the SS7 protocol. This protocol allows the message transfer of messages not bigger than 144 characters. Due to this restriction in size, it is unlikely to be suitable for this type of communication as the keys being required are already sized similarly, leaving no space for Messages or routing information.

The 3rd Generation Partnership Project (3GPP) maintains the Multimedia Messaging Service (MMS). This protocol is mainly a mobile protocol based on telephone networks.

Both protocols are not widely adopted within the Internet domain. There are gateways providing bridging functionalities to the SMS/MMS services. However, the protocol itself is insignificant on the Internet itself.

B.3.11 MMS

This protocol is just like the SMS protocol accessible through the Internet by using gateways but not directly usable within the Internet.

B.4 Results

We have shown that all common M2M protocols failed mainly at Ct3 as there is no need for message routing. In M2M communication contacting foreign machines is not common. Therefore M2M protocolls are typically using static M2M communication over prepared channels. Such behavior is, however unsuitable for a generic messaging protocol.

Pure storage protocols fail at the same criteria as they typically have a defined set of data sources and data sinks, whereas usually at least the data sources are limited in number. This makes those protocols unsuitable again.

We can clearly state that according to the criteria, only a few protocols are suitable. Table B.1 on page 12 shows that only SMTP and XMPP are suitable protocols. Eventually, similar protocols such as HTTP (with WebDAV) or FTP may be usable as well.

Protocol \ Criteria	Ct1: Widely adopted	Ct2: Reliable	Ct3: Symmetrically built
HTTP	✓	✓	×
FTP	✓	✓	×
TFTP	×	×	×
MQTT	~	✓	×
AMQP	~	✓	×
CoAP	~	~	×
WAMP	×	✓	~
XMPP	✓	✓	✓
SMTP	✓	✓	✓

Table B.1: comparison of protocols in terms of the suitability as transport layer

The findings of this short analysis suggested that we should use the two protocols, SMTP and XMPP, for our first standardization. We require at least two to prove that the protocol is agnostic to the transport.

C Glossary

adversary In this work, we are referring to an adverser to any entity opposing to the privacy of a message. For a more throughout definition refer to section 4.1

anonymity We refer to the term anonymity as defined in [**anonTerminology**]. “Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set.”[1]

Sender Anonymity The anonymity set is the set of all possible subjects. For actors, the anonymity set consists of the subjects who might cause an action. For actees, the anonymity set consists of the subjects which might be acted upon. Therefore, a sender may be anonymous (sender anonymity) only within a set of potential senders, his/her sender anonymity set, which itself may be a subset of all subjects worldwide who may send a message from time to time.

Receiver Anonymity The same for the recipient means that a recipient may be anonymous (recipient anonymity) only within a set of potential recipients, his/her recipient anonymity set. Both anonymity sets may be disjoint, be the same, or they may overlap. The anonymity sets may vary over time.

Agent An agent is a single component of a service (Service) provided to a user or other services.

eID An ephemeral identity (eID) is a unique user of a vortex node characterized by its public key. This user is created with a VortexMessage and has only a limited lifetime. After expiry all informations related to this identity are deleted.

EWS Exchange Web Services (EWS) are a Microsoft proprietary protocol to access exchange services from a client. It may be regarded as an alternative to IMAPv4. This is however incomplete as EWS offers additional features such as User Configuration, Delegate Management or Unified Messaging.

IMAP IMAP (currently IMAPv4) is a typical protocol to be used between a Client MRA and a Remote MDA. It has been specified in its current version in [**RFC3501**]. The protocol is capable of fully maintaining a server-based message store. This includes the capability of adding, modifying, and deleting messages and folders of a mailstore. It does not include however sending mails to other destinations outside the server-based store.

Item of Interest (IoI) The Item of Intrest (IoI) are defined in [**anonTerminology**] and refer to any subject action or entity which is of interest to a potential adversary.

LMTP The Local Mail Transfer Protocol is defined in [**RFC2033**]. This RFC defines a protocol similar to SMTP for local mail senders. This protocol allows a sender to have no mail queue at all and thus simplifies the client implementation.

Local Mail Store A Local Mail Store offers a persistent store on a local non-volatile memory in which messages are being stored. A store may be flat or structured (e.g., supports folders). A local mail store may be an authoritative store for mails or a “cache only” copy. It is typically not a queue.

Server Admin We do regard a server admin as a person with high privileges and profound technical knowledge of a server and its associated technology. A Server Admin may have access to one or multiple servers of the same kind.

MDA An MDA provides uniform access to a local message store.

Remote MDA A Remote MDA is typically supporting a specific access protocol to access the data stored within a local message store.

Local MDA A Local MDA is typically giving local applications access to a server store. This may be done thru an API, a named socket or similar mechanisms.

MRA A Mail Receiving Agent is an agent, which receives emails from another agent. Depending on the used protocol two subtypes of MRAs are available.

Client MRA A client MRA picks up emails in the server mail storage from a remote MDA. Client MRAs usually connect through a standard protocol that was designed for client access. Examples for such protocols are POP or IMAP.

Server MRA Unlike a Client MRA, a server MRA listens passively for incoming connections and forwards received messages to an MTA for delivery and routing. A typical protocol supported by a server MRA is SMTP

MS-OXCMAPIHTTP Microsofts Messaging Application Programming Interface (MAPI) Extensions for HTTP specifies the Messaging Application Programming Interface (MAPI) Extensions for HTTP in [**ms-oxcmapihttp**], which enable a client to access personal messaging and directory data on a server by sending HTTP requests and receiving responses returned on the same HTTP connection. This protocol extends HTTP and HTTPS.

MSA A Mail Sending Agent. This agent sends emails to a Server MRA.

MTA A Mail Transfer Agent. This transfer agent routes emails between other components. Typically an MTA receives emails from an MRA and forwards them to an MDA or MSA. The main task of an MTA is to provide reliable queues and solid track of all emails as long as they are not forwarded to another MTA or local storage.

MTS A Mail Transfer Service. This is a set of agents which provide the functionality to send and receive messages and forward them to a local or remote store.

MSS A Mail Storage Service. This is a set of agents providing a reliable store for local mail accounts. It also provides Interfacing which enables clients to access the users’ mail.

MUA A Mail User Agent. This user-agent reads emails from local storage and allows a user to read existing emails, create and modify emails.

MURB A multi use reply block. This tzpe of routing block is provided by a sender to give a node the possibilitz to route back answers without the knowledge of a location of the sender. In contrast to a SURB a MURB may be used mutiple times. The number of times is regulated by the $maxReplay$ field. Furthermore a MURB must provide multiple peer keys for all routing steps to avoid repeating patterns of key blocks. This structure makes a MURB much bigger than a SURB.

Privacy From the Oxford English Dictionary: “

1. The state or condition of being withdrawn from the society of others, or from the public interest; seclusion. The state or condition of being alone, undisturbed, or free from public attention, as a matter of choice or right; freedom from interference or intrusion.
2. Private or retired place; private apartments; places of retreat.
3. Absence or avoidance of publicity or display; a condition approaching to secrecy or concealment. Keeping of a secret.
4. A private matter, a secret; private or personal matters or relations; The private parts.
5. Intimacy, confidential relations.
6. The state of being privy to some act.

”[**OXFORD**]
In this work, privacy is related to definition two. Mails should be able to be handled as a virtual private place where no one knows who is talking to whom and about what or how frequent (except for directly involved people).

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

> A pseudonym is an identifier of a subject other than one of the subject’s real names. The subject which the pseudonym refers to is the holder of the pseudonym. A

[1] footnotes omitted in quote

> subject is pseudonymous if a pseudonym is used as an identifier instead of one of its real names.[2]

POP POP (currently in version 3) is a typical protocol to be used between a Client MRA and a Remote MDA. Unlike IMAP, it is not able to maintain a mail store. Its sole purpose is to fetch and delete emails in a server-based store. Modifying Mails or even handling a complex folder structure is not doable with POP

RBB A routing block builder (RBB) is a VortexNode assembling the operations and hops for a message. If the RBB is not equal to the sender of the message the receiver may be anonymous to the sender.

Service A service is an endpoint on a server providing the functionality to a client. This service may consist of several Agents (Agent).

SMTP SMTP is the most commonly used protocol for sending emails across the Internet. In its current version it has been specified in [**RFC5321**].

Storage A store to keep data. It is assumed to be temporary or persistent.

SURB A single use reply block. This type of routing block is provided by a sender to give a node the possibility to route back answers without the knowledge of a location of the sender. A SURB may only be used once subsequent uses of the block are not possible. The lifetime of a SURB is typically limited to minutes or hours.

UBM We use the term Unsolicited Bulk Message as a term for any mass message being received by a user without prior explicit consent. A less formal term for such a message in email terminology is spam or junk mail.

Undetectability As undetectability we take the definition as specified in [**anonTerminology**].

> Undetectability of an item of interest (IOI) from an attacker's perspective means that the attacker cannot sufficiently distinguish whether it exists or not.[2]

Unlikability We refer to the term unlinkability as defined in [**anonTerminology**]. "Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not.

Unobservability As unobservability we take the definition as specified in [**anonTerminology**].

> Unobservability of an item of interest (IOI) means
>
> - undetectability of the IOI against all subjects uninvolved in it and
> - anonymity of the subject(s) involved in the IOI even against the other subject(s) involved in that IOI.

As mentioned in this paper, unobservability raises the bar of required attributes again ($\Rightarrow$ reads "implies"):

$$\begin{aligned} censorship\ resistance &\Rightarrow unobservability \\ unobserability &\Rightarrow undetectability \\ unobserability &\Rightarrow anonymity \end{aligned}$$

user Any human or technical entity using a system not following strict message processing rules. A user does always interface a non-interface related entity and is triggered by this or triggers it related, but not limited, to the message.

XMPP The Extensible Messaging and Presence Protocol (XMPP)[**RFC6120**, **RFC6121**] was formerly also known as Jabber protocol. It is an extensible instant messenger protocol widely adopted in chat clients.

Zero Trust Zero trust is not a truly researched model in systems engineering. It is, however, widely adopted. We refer in this work to the zero trust model when denying the trust in any infrastructure not directly controlled by the sending or receiving entity. This distrust extends especially but not exclusively to the network transporting the message, the nodes storing and forwarding messages, the backup taken from any system except the client machines of the sending and receiving parties, and software, hardware, and operators of all systems not explicitly trusted. As explicitly trusted in our model, we do regard the user sending a message (and his immediate hardware used for sending the message), and the users receiving the messages. Trust in between the receiving parties (if more than one) of a message is not necessarily given.

[2] footnotes omitted in quote

C Bibliography

C 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.

,