



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

July 9, 2020

Original document available on the edoc sever of the university of Basel edoc.unibas.ch.



5 This work is published under "Creative Commons Attribution-NonCommercial-NoDerivatives 3.0 Switzerland" (CC BY-NC-ND 3.0 CH) licensed. The full license can be found at <http://creativecommons.org/licenses/by-nc-nd/3.0/ch/>.

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 L^AT_EX project for free. It is due to their efforts that we can generate professionally typeset PDFs (and far more) for free.

Contents

I Introduction	1
1 Preface	3
2 Our Contribution	5
3 Scope and Approach	6
4 Notation	7
4.1 Cryptography	7
4.2 Code and commands	8
4.3 Hyperlinking	8
5 Document Structure and Systematics	8
II Relevant Concepts and Technologies	9
6 Anonymity Research	11
6.1 Definition of Anonymity	11
6.2 k -Anonymity	12
6.3 ℓ -Diversity	12
6.4 t -Closeness	12
6.5 Single Use Reply Blocks and Multi-Use Reply Blocks	13
6.6 Censorship	13
6.6.1 Censorship Resistant	13
6.6.2 Parrot Circumvention	14
6.6.3 Censorship Circumvention	14
7 Related Cryptographic Theory and Algorithms	15
7.1 Deniable Encryption and Deniable Steganography	17
7.2 Key Sizes	17
7.3 Cipher Mode	17
7.4 Summary of Cipher Modes	20
7.5 Padding	20
7.6 Pseudo Random Number Generators	21
8 Censorship Circumvention	21
8.1 Covert Channel and Channel Exploitations	21
8.2 Steganography	22
8.3 Timing Channels	22
8.4 Technical Forms of Censorship	22

8.4.1	Making Systems Unavailable by Censoring Lookups	22
8.4.2	Making Systems Unavailable by Disrupting System Traffic	23
8.4.3	Making Systems Unavailable by Interfering with System Traffic	23
8.5	Spread Spectrum in Networking Protocols	23

9	Other Related Concepts	23
9.1	Zero Trust	23

III Anonymous Communication Systems 25

10	Well Known Standard Protocols	27
-----------	--	-----------

11	Information Routing and Distribution for Anonymizing Protocols	30
11.1	Mixing	31
11.2	Anonymous Remailers	31
11.3	Onion Routing	32
11.4	Garlic Routing	32
11.5	Crowds	32
11.6	Mimic Routes	33
11.7	Distributed Hash Tables	33
11.8	Dining Cryptographer Networks	33

12	Proposed Academic Protocols and System Implementations	33
-----------	---	-----------

12.1	Characteristics of Known Anonymity Implementations	33
12.2	Resenders, Onion Routers, and MixNets	34
12.2.1	Pseudonymous Remailers (1981)	34
12.2.2	Cyberpunk Remailers (approx. 1993)	34
12.2.3	Babel (1996)	35
12.2.4	Mixmaster-Remailers (1996)	35
12.2.5	Crowds (1997)	35
12.2.6	Tor (2000)	35
12.2.7	I^2P (2001)	36
12.2.8	Mixminion-Remailers (2002)	37
12.2.9	\mathcal{P}^5 (2002)	37
12.2.10	AN.ON (2003)	37
12.2.11	AP3 (2004)	37
12.2.12	Cashmere (2005)	38
12.2.13	SOR (2012)	38
12.2.14	SCION (2017)	38
12.3	Distributed Hash Tables	38
12.3.1	Tarzan (2002)	38
12.3.2	MorphMix (2002)	38
12.3.3	Salsa (2008)	39
12.4	Dining Cryptographer Based Network	39
12.4.1	Herbivore (2003)	39
12.4.2	Dissent (2010)	39
12.5	Broadcast and Multicast Networks	39
12.5.1	Hordes (2002)	39
12.6	Distributed Storage Systems	39
12.6.1	Feenet (2000)	39
12.6.2	Gnutella (2000)	40
12.6.3	Gnutella2 (2002)	40

12.7	Unknown (TBD)	40
12.7.1	Riffle (2016)	40
12.7.2	Atom (2016)	40
12.7.3	Riposte (2015)	40
12.7.4	Pung (2016)	40
12.7.5	PIR (2018)	41
12.7.6	Karaoke (2018)	41
12.7.7	Loopix (2017)	41
12.7.8	Stadium (2017)	41
12.7.9	Vuvuzela (2015)	41

IV The MessageVortex System 43

13 Requirements for an Anonymizing Protocol	45
13.1 Threat Model	45
13.1.1 Observing Adversaries	46
13.1.2 Censoring Adversaries	47
13.2 Required Properties for our Unobservable Protocol	47
13.2.1 System requirements	47
13.2.2 Message Requirements	49
13.2.3 Operational Requirements	50

14 Rationale	51
14.1 System Design and Infrastructure	51
14.2 Message and Routing	52
14.3 Summarizing Chosen Approaches for <i>MessageVortex</i>	53

15 Protocol	55
15.1 Protocol Terminology	55
15.2 Key Components	56
15.2.1 Nodes and their identities	56
15.2.2 Workspaces and Ephemeral Identities	57
15.2.3 Protocol Layers	57
15.2.4 Transport Layer	58
15.2.5 VortexMessages	62
15.2.6 Routing Operations	64
15.3 Summary	68

V Implementation 69

16 Selection of Algorithms, Encodings, and Protocols	71
16.1 Encoding Scheme	72
16.2 Cipher Selection	72
16.3 Mode Selections	74
16.4 Padding selection	76
16.4.1 RSAES-PKCS1-v1_5 and RSAES-OAEP	76
16.4.2 PKCS7	76
16.4.3 OAEP with SHA and MGF1 padding	76
16.4.4 Honorable Mention: AP Padding for <i>redundancy</i> Operations	76

16.4.5	Pseudo Random Number Generator Selection	77
16.5	Transport Layer Protocol Selection	77
16.5.1	Applied Criteria	78
16.5.2	Analyzed Protocols	78
16.5.3	Analysis	79
16.5.4	Results	83
17	Transport Layer Implementation	83
17.1	Implementation of a Dummy Transport Layer	83
17.2	Implementation of an Email Transport Layer	83
17.3	Implementation of an XMPP Transport Layer	84
17.4	Distributed Configuration and Runtime Store of processing content	84
18	Blending Layer Implementation	85
18.1	Embedding Spec	85
18.1.1	Plain Embedding	85
18.1.2	Implementation of F5 Blending	85
18.2	Blending Content Generation	85
19	Routing Layer Implementation	85
19.1	ASN.1 encoding scheme	86
19.2	Processing of messages	86
19.2.1	Processing of Incoming Messages	86
19.2.2	Processing of Outgoing Messages	86
19.2.3	Implementation of Operations	86
19.3	Request handling	86
19.3.1	Requesting a new Ephemeral Identity	86
19.3.2	Replacing an Existing Node Identity	87
19.3.3	Replacing an Existing Reply Block	87
20	Accounting Layer Implementation	87
21	Usability Related Implementation Details	87
21.1	Addressing and address representations	88
21.2	Linking to Common User Agents	88
22	Efficiency Related Implementation Details	88
22.1	Node Storage Management	88
22.1.1	Life-cycle of Ephemeral Identities and Workspaces	88
22.1.2	Life-cycle of Requests	88
22.1.3	Life-cycle of Operations	88
22.2	Overhead Minimization in Message	88
VI	Operational concerns	89
23	General Concerns Regarding Operation	91
23.1	Hardware	91
23.2	Addressing of Vortex Nodes	91
23.3	Client	91
23.3.1	Vortex Accounts	92
23.3.2	Vortex Node Types	92
23.4	Strategies for Composing Routing Blocks	93

23.4.1 MURBs	93
23.5 Algorithms Suitable for Achieving Anonymity	93
23.6 Possibilities of Routing Diagnosis and Reputation Building	94
23.7 Possibilities of Redundancies	94
24 Protocol Bootstrapping	94
24.1 Key Distribution for Endpoints	94
24.2 Key Aquisition for Routing Nodes	94
25 Operational Concerns Regarding Environments with an Observing Adversary	94
26 Operational Concerns Regarding Environments with an Censoring Adversary	94
27 Real World Problems when using <i>MessageVortex</i>	94
27.1 Size Restrictions of the Transport Layer	94
VII Analysis of MessageVortex	95
28 Analysis of the Blending Layer	97
29 Analysis of the Core Operations	97
29.1 Splitting and Merging	97
29.2 Encryption and Decryption Operations	97
29.3 Add and Remove Redundancy Operations	97
30 Well Known Attacks	97
30.1 Broken Encryption Algorithms	97
30.2 Attacks Targeting Anonymity	98
30.2.1 Denial of Service Attacks	101
31 Identification of Possible Attack Schemes and Mitigation	102
31.1 Static Attacks	102
31.1.1 Bugging and Tagging Attacks	102
31.1.2 Information Leaking related to Information Available to Routing Nodes	102
31.1.3 Identification of involved Nodes	102
31.1.4 Identification of MessageVortex Traffic	102
31.2 Dynamic Attacks	103
31.2.1 Attacks against the vortex system itself	103
31.2.2 Attacking a single ephemeral Identity of a MessageVortex Node	103
31.2.3 Attacking Sending and Receiving Identities of the MessageVortex System	103
31.2.4 Recovery of Previously Carried Out Operations	103
32 Analysis of the effectiveness of Attack Schemes	103
33 Analysis of the Degree of Anonymization in Comparison to other Systems	103
34 To be placed (TBP)	103
34.1 Side Channel Leaking	103
34.1.1 Software Updates and Related Data Streams	103
34.1.2 Bugging in transported messages	103

VIII Discussion on Results	105
35 Measuring up to the Requirements	107
36 Achieved level of anonymity	107
37 Weaknesses of the protocol	107
38 Further and Missing Research	107
IX Appendix	109
A The RFC draft document	A1
B Glossary	A60
Bibliography	A65
Short Biography	A66

List of Corrections

Warning: Rewrite core question	6
Warning: reference respective sections	6
Warning: reference respective sections	7
Warning: reference respective sections	7
Warning: Insert overview over preexisting work	7
Warning: Write last	8
Warning: complete section	11
Warning: Extend PRNG section. Where is the rest?	21
Warning: Explain further the problem for people not so much in cloud and networks	23
Warning: Explain more and maybe add china as an example	23
Warning: Empty section	23
Warning: Empty section	23
Warning: incomplete section	24
Warning: complete section	30
Warning: complete section	33
Warning: complete section	33
Warning: Add section	35
Warning: Add Riffle	40
Warning: Add Atom	40
Warning: Add Riposte	40
Warning: Add Pung	41
Warning: Check PIR	41
Warning: Extend Karaoke and add to comparison	41
Warning: Add Loopix	41
Warning: Add Stadium	41
Warning: Sumarize subsections	71
Warning: Story: Both embeddings allow unbroken logical structure (plain->generic and F5 for JPEG/GIF/BMP). When advertizing multiple offsets or PWs then all of them are checked by blending layer. Presence of a valid header block is sufficient proof.	85
Warning: complete section with story "Implementation of F5 Blending"	85
Warning: Story: F5 [f5] is very old and has not been broken. The first implementation had a detectable issue with an unintended double compression [steganalysisf5] but the authors of f5 fixed this issue [F5broken]. newer derivates and alternatives are available (e.g., nsF5 or MSET)	85
Warning: complete section	85
Warning: complete section	85
Warning: incomplete section	86
Warning: complete section	86
Warning: complete section	86
Warning: complete section	86
Warning: Mention mapping operation	86
Warning: Mention floating point issues when splitting	86
Warning: complete section	86
Warning: complete section	86
Warning: complete section	86
Warning: complete section	87
Warning: complete section	87
Warning: complete section	87
Warning: complete section	88
Warning: complete section	88

Warning: complete section	88
Warning: part is highly incomplete	91
Warning: complete section	93
Warning: complete section	94
Warning: insert operational	94
Warning: Fill in results of the RS analysis regarding the findings of the protocol	97
Warning: complete section	102
Warning: complete section	103
Warning: complete section	107

Introduction

*The most effective way to do it is TO
DO IT
Amelia Earhart*

1 Preface

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 18 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.

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)[**iccppr**].

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 1st, 2016 Countries such as China (signatory), Cuba (signatory), Qatar, Saudi Arabia, Singapore, United Arab Emi-

rates, 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., BGP, 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 be 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 analyzes 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 1.1: 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 to an adversary.

2 Our Contribution

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

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

To non-traceable routing this, we introduce a novel type of routing operation called “addRedundancy”. This operation is a Reed-Solomon-calculation with encryption and a new type of padding. This op-

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

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

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

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

3 Scope and Approach

The main topic of this thesis was defined as follows:



- Is it possible to have a messaging protocol used on the Internet, based on “state of the science” technologies offering a high level of unlinkability (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?

This question covers the principal part of the work. We first elaborate on a list of criteria for the *MessageVortexprotocol*. 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.



2. How can entities utilizing *MessageVortex* be attacked, and what measures are available to circumvent such attacks?

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).



3. How can design mitigate attacks target anonymity of a sending or receiving entity within *MessageVortex*?

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.



4 Notation

4.1 Cryptography

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

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 asymmetric 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 $\mathbf{M}\langle\mathbf{t}, \mathbf{MURB}, \mathbf{serial}\rangle$ respectively if encrypted $\mathbf{M}^{K_a}\langle\mathbf{t}, \mathbf{MURB}, \mathbf{serial}\rangle$.

asymmetric: $E^{K_a^{-1}}(\mathbf{M})$	$= \mathbf{M}^{K_a^{-1}}$
$D^{K_a^1}(E^{K_a^{-1}}(\mathbf{M}))$	$= \mathbf{M}$
$D^{K_a^{-1}}(E^{K_a^1}(\mathbf{M}))$	$= \mathbf{M}$
symmetric: $E^{K_a}(\mathbf{M})$	$= \mathbf{M}^{K_a}$
$D^{K_a}(E^{K_a}(\mathbf{M}))$	$= \mathbf{M}$
hashing (unsalted): $H(\mathbf{M})$	$= \mathbf{H}_M$
hashing (salted): $H^{S_a}(\mathbf{M})$	$= \mathbf{H}_M^{S_a}$

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:

Prefix component: PREFIX	$= D^{K_a^1}(\mathbf{P}^{K_a^{-1}}) = D(\mathbf{P})$
Header component: HEAD	$= D^{K_a^1}(\mathbf{H}^{K_a^{-1}}) = D(\mathbf{H})$
Route component: ROUTE	$= D^{K_a^1}(\mathbf{R}^{K_a^{-1}}) = D(\mathbf{R})$

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

4.2 Code and commands

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

```
1 public class Hello {  
2     public static void main(String args[]) {  
3         System.out.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 “ \leftarrow ” is inserted to indicate that a line break was inserted for readability.

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

4.3 Hyperlinking

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

5 Document Structure and Systematics



Relevant Concepts and Technologies

*Where does a snake's tail start?
My Son*



6 Anonymity Research

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

6.1 Definition of Anonymity

As the definition for Anonymity we take the definition as specified in [[anonTerminology](#)].

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

and

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

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

6.2 k -Anonymity

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.

6.3 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.

6.4 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.

¹footnotes omitted in quote

6.5 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.

6.6 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.

6.6.1 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.

6.6.2 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. 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.

6.6.3 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.

6.6.3.1 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.

6.6.3.2 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 ?? on page ??). We limited our search to algorithms capable of hiding binary data within these files. The number of academically researched options was surprisingly low.

After reviewing the options, we decided to go for F5[f5]. It is a reasonably well-researched algorithm which attracted many researchers. The original F5 implementation had a detectable issue with

artifacts [F5broken] caused by the recompression of the image. This issue was caused only due to a problem in the reference implementation, and the researchers have provided a corrected reference implementation without the weakness.

YASS, as described in [solanki2007yass], was not considered a candidate. Although less researched, researchers found multiple weaknesses [kodovsky2010modern, li2009steganalysis].

6.6.3.3 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.

7 Related Cryptographic Theory and Algorithms

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

$$D^{K_a}(E^{K_a}(\mathbf{M})) = \mathbf{M} \quad (7.1)$$

For a key $K_b \neq K_a$ this means

$$D^{K_a}(E^{K_b}(\mathbf{M})) \neq \mathbf{M} \quad (7.2)$$

$$D^{K_b}(E^{K_a}(\mathbf{M})) \neq \mathbf{M} \quad (7.3)$$

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...

$$D^{K_a^{-1}}(E^{K_a^1}(\mathbf{M})) = \mathbf{M} \quad (7.4)$$

$$D^{K_a^1}(E^{K_a^{-1}}(\mathbf{M})) = \mathbf{M} \quad (7.5)$$

It is important that

$$D^{K_a^{-1}}(E^{K_a^{-1}}(\mathbf{M})) \neq \mathbf{M} \quad (7.6)$$

$$D^{K_a^1}(E^{K_a^1}(\mathbf{M})) \neq \mathbf{M} \quad (7.7)$$

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

$$D^{K_b^{-1}}(E^{K_a^1}(\mathbf{M})) \neq \mathbf{M} \quad (7.8)$$

$$D^{K_b^1}(E^{K_a^1}(\mathbf{M})) \neq \mathbf{M} \quad (7.9)$$

$$D^{K_b^{-1}}(E^{K_a^{-1}}(\mathbf{M})) \neq \mathbf{M} \quad (7.10)$$

$$D^{K_b^1}(E^{K_a^{-1}}(\mathbf{M})) \neq \mathbf{M} \quad (7.11)$$

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 known 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 7.2 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.

- McEliece

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 ¹ 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.

- ElGamal

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

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.

7.1 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.

7.2 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 ≈ 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.

7.3 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: $H_{GFmult}(o \oplus (len(A)||len(B))) \oplus E^{K^0}(counter_0)$. 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 ECB 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.

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

Table 7.1: comparison of encryption modes in terms of the suitability

7.4 Summary of Cipher Modes

7.5 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.

7.5.0.1 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 padding may give a clue if decryption was successful or not. RSAES-OAEP is the newer of the two padding standards

7.5.0.2 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.

7.5.0.3 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 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

¹included in auth

²Requires tweak instead of IV

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.

7.6 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 in some of the later specified operations.

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 Mical PRNG. Both PRNGs were quoted to pass BigCrush. However, recent development shows that this might not be true for xorshift128+, as demonstrated in [LEMIRE2019139].



8 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.

8.1 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.

8.2 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 ?? on page ??). We limited our search to algorithms capable of hiding binary data within these files. The number of academically researched options was surprisingly low.

After reviewing the options, we decided to go for F5[f5]. It is a reasonably well-researched algorithm which attracted many researchers. The original F5 implementation had a detectable issue with artifacts[F5broken] caused by the recompression of the image. This issue was caused only due to a problem in the reference implementation, and the researchers have provided a corrected reference implementation without the weakness.

YASS, as described in [solanki2007yass], was not considered a candidate. Although less researched, researchers found multiple weaknesses[kodovsky2010modern, li2009steganalysis].

8.3 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.

8.4 Technical Forms of Censorship

There are many types of censorship available within technical systems. An in-depth understanding of the possibilities is required in order to understand the means of a censoring adversary.

8.4.1 Making Systems Unavailable by Censoring Lookups

This is one of the cheapest method to create censorship. Lookup systems such as DNS servers are modified in such a way that traffic is no longer deliverable or redirected to a system controlled by the censorer.

Many jurisdictions have implemented such measures a very cheap measure of censorship. It is, however, very easy to outcurve. As soon as a user no longer uses adversary controlled lookup services, this form of censorship is ineffective. In the case of DNS this means either:

- Use a public DNS server worldwide available
- Use another protocol to hide the traffic

- A legit protocol with tunnelling capabilities like SSH to a system outside of the reach of the censoring adversary.
- Use a fully blown tunnel such as a VPN.
- Piggyback a legit protocol such as DNS-over-HTTPS (DoH)[[rfc8484](#)] or DNS-over-XMPP[[xep0418](#)]

8.4.2 Making Systems Unavailable by Disrupting System Traffic

Disruption of traffic is commonly done with packet filtering devices commonly referred as firewalls. These firewalls may filter any traffic to a given system. There are some considerable downsides to this system from the adversary's point of view.

First, it requires high bandwidth. All traffic of a jurisdiction or target must pass through this filtering device. This is usually doable for a country.

Secondly, the target must be identifiable on a technical level (e.g., IP address) as content based filtering is only feasible with unencrypted or weakly protected systems. This technical identification is hard as systems may change their addresses dynamically either due to cloud related elasticity or simply due to an incomplete view to a distributed system.



And lastly, The target may not be split (e.g., partial censoring). Censoring platforms such as Youtube or Gmail which are commonly shared among legit and illegit uses from the censors point-of-view can not be reasonably split. This due to the fact that only the provider of the service can do selective censoring on the system.



8.4.3 Making Systems Unavailable by Interfering with System Traffic



8.5 Spread Spectrum in Networking Protocols



9 Other Related Concepts

9.1 Zero Trust

Zero trust is not an academically defined concept. In fact it is widely misused by many marketing departments of well known devices and applications related to security.

We however define zero trust as follows:

We assume that any participant in a network not explicitly trusted may:

- Make some or all information available to him available to others.
- Analyze all information within his reach.
- Willingly break protocol rules in order to gain information or other advantages.

As design principle this means that information is kept hidden as much as possible within the system.



Anonymous Communication Systems

It was the anonymity. He wanted to be unknown, unpossessed by others' knowledge of him. That was freedom.

Ling Ma, Severance

10 Well Known Standard Protocols

SMTP and Related Post Office Protocols (1982)

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

To understand the routing of a mail, it is essential to understand the whole chain starting from a user(-agent) until arriving at the target user (and being read!). To simplify this, we used a consistent model that includes all components (server and clients). The figure 10.1 shows all involved parties of a typical mail routing. It is essential to understand that mail routing remains the same regardless of the client. However, the availability of a mail at its destination changes drastically depending on the type of client used. Furthermore, control of the mail flow and control is different depending on the client.

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

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

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

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

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

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

An MSS consists of local storage and delivery agents which do offer uniform interfaces to access the local store. They do also deal with replication issues, and grant should take care of the atomicity of transactions committed to the storage. Typically there are two different kinds of MDAs. Local MDAs

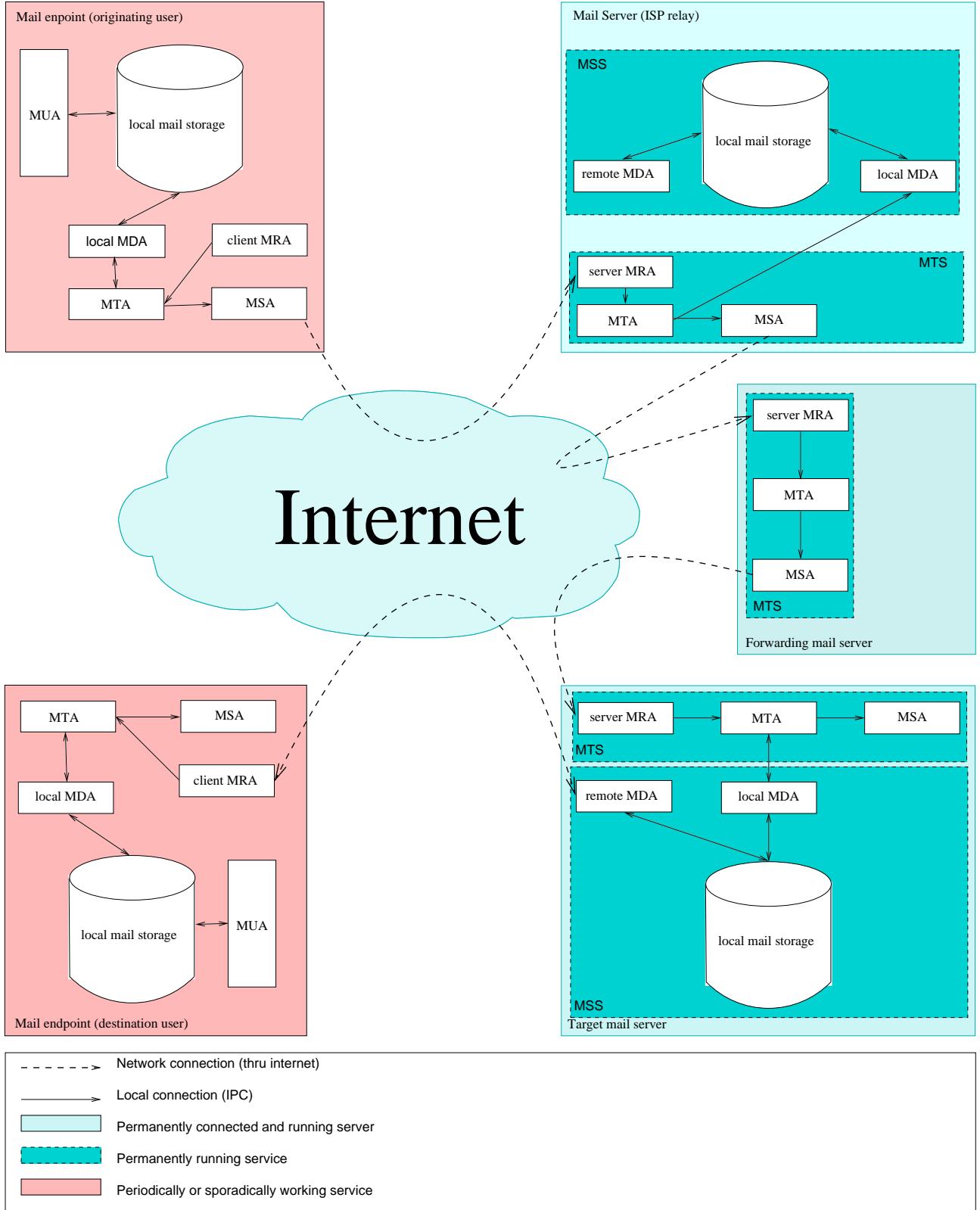


Figure 10.1: Mail Agents

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.

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%). [\[clientDistribution2\]](#) as a more recent source lists in February 2014 iOS devices with 37%, followed by Outlook (13%), and Google Android (9%).

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.

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.

S/MIME (1996)

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

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

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

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

Pretty Good Privacy (1996)

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

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

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

11 Information Routing and Distribution for Anonymizing Protocols



11.1 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.

11.2 Anonymous Remailers

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

Pseudonymous Remailers (also called Nym Servers) take a message and replace all information pointing to the original sender with a pseudonym. This pseudonym may be used as an answer address. The most well known pseudonymous remailer possibly was [anon.penit.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 12.2.1

Cypherpunk remailers forward messages like pseudonymous remailers. Unlike pseudonymous remailers, Cypherpunk remailers decrypt a received message, and its content is forwarded without adding a pseudonym. A reply to such a message is not possible. They may, therefore, be regarded as an “decrypting reflector” or a “decrypting mix” and may be used to build an onion routing network for messages. For a more in-depth discussion of type-1-remailers, see section 12.2.2.

Mixmaster remailers are very similar to Cypherpunk remailers. Unlike them, Mixmaster remailers hide the messages, not in an own protocol, but use SMTP instead. While using SMTP as a transport layer, Cypherpunk remailers are custom (non-traditional mail) servers listening on port 25. For a more in-depth discussion of type-2-remailers, see section 12.2.4.

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

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

For a more in depth discussion of Mixminion remailers see section 12.2.8.

11.3 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.

Onionized messages typically have the problem of a constant size loss throughout the system. Some systems counter this effect, by separating the routing setup from the message path.

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 12.2.6.

11.4 Garlic Routing

Garlic routing is an improved form of onion routing. To stop onionized messages to continuously loose contents through their way, a garlic router collects multiple, independent messages into one message before routing. This compensates for the “size loss effect” of onionized systems.

11.5 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:itissec] 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 [crowdsAttack], 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.

11.6 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.

11.7 Distributed Hash Tables



11.8 Dining Cryptographer Networks

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

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

12 Proposed Academic Protocols and System Implementations



12.1 Characteristics of Known Anonymity Implementations

Table ?? shows the previously analyzed protocols.

	Network Structure					Routing Information	Communication Model				Performance and Deployability				
	Topology	Connection		Symmetry			Network view	Updating	Routing Type	Node Selection			Latency	Communication mode	
		Direction	Synchronization	Roles	Hierarchy					Scheduling	Determinism	Selection set			
Message Vortex	☒	↔	☒	○	●	↳	...	✗	○	⊗	✗	✓	✓
Chaum Mixes ¹	☒	→	≠	✗	●	✗	...	✓	⊗	○	✗	✓	✗
Babel ¹	☒	→	≠	○	●	✗	...	✓	⊗	○	✗	✗	✗
Mixmaster ¹	☒	→	≠	○	●	✗	...	✗	⊗	○	✗	✓	✓
Mixminion ¹	☒	→	≠	○	●	⊕	...	✗	⊗	○	✗	✓	✓
Tor ¹	□	↔	☒	○	●	✗	⊗	○	✗	✓	✗
Crowds ¹	☒	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✗
MorphMix ¹	□	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✓
Tarzan ¹	□	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✓
Torsk ¹	□	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✓
AP3 ¹	□	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✗
Salsa ¹	□	↔	≠	○	●	↳	...	✗	⊗	○	✗	✓	✗
Chaum's DCnet ¹	☒	→	≠	✗	●	↳	...	✓	⊗	○	✗	✗	✗
Herbivore ¹	□	→	≠	...	⊕	○	●	↳	...	✓	⊗	○	✗	✓	✗
Dissent in numbers ¹	□	→	≠	..	⊕	○	●	↳	...	✓	⊗	○	✗	✓	✗
I ² P ¹	□	→	≠	○	●	⊕	...	✗	⊗	○	✗	✓	✗
P ⁵ ¹	□	→	≠	...	⊕	○	●	↳	...	✓	⊗	○	✗	✓	✗
Riffle															
Atom															
Riposte															
Pung															
PIR															
Karaoke															
Loopix															
Stadium															
Vuvuzela															

Table 12.1: Classification table for anonymization protocols according to [Shirazi2018]

12.2 Resenders, Onion Routers, and MixNets

12.2.1 Pseudonymous Remailers (1981)

A pseudonymous remailer allows to reach people via a pseudonymous email address. On the remailing server all traces of the original sender are removed and a pseudonymous email is inserted instead. The foundation of these remailers can be found in an early article of David Chaum[CHAUM1].

One of the most famous remailers was the Penet remailer (anon.penet.fi). This remailer only lasted from 1993 to 1996 and was shut down after at least two compromises involving the Chruch of Scientology. Details of the closure can be found in [penetClosure].

12.2.2 Cypherpunk Remailers (approx. 1993)

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

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

12.2.3 Babel (1996)

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

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

12.2.4 Mixmaster-Rmailers (1996)

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

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

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

12.2.5 Crowds (1997)



12.2.6 Tor (2000)

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

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

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

Many attacks involving the Tor networks have been discussed in the academic world such as **[hs-attack06, esorics13-cellflood, bauer:wpes2007, esorics12-torscan, oakland2013-trawling, danner-et-al:itissec12, 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 Kazakhstan. 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.

12.2.7 *I²P* (2001)

The name *I²P* is derived from “Invisible Internet Project” according to geti2p.net. The first binary release on sourceforge dates from 2001. The system itself is comparable to Tor for its capabilities. Major differences are:

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

I²P 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 *I²P*s security model is chosen based on IP addresses, the authors propose to use several cloud providers in different B-Class networks. By selectively flooding peers, an adversary may extract statistical information. The paper proposes an attack based on the heuristic performance-based peer selection. The main critics of the paper were that the peer selection might be influenced by an adversary enabling him to recover *I²P* 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 *I²P*s security model is chosen based on IP addresses, the authors propose to use several cloud providers in different B-Class networks. By selectively flooding peers, an adversary may extract statistical information. The

paper proposes an attack based on the heuristic performance-based peer selection. The main critics of the paper were that the peer selection might be influenced by an adversary enabling him to recover data on a statistical base.

12.2.8 Mixminion-Rmailers (2002)

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

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

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

12.2.9 \mathcal{P}^5 (2002)

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

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

12.2.10 AN.ON (2003)

AN.ON, as suggested in [**fedorath2003system**], is a mixing network. It generates messages in equally sized chunks and sends them in fixed time slots after random mixing. Its implementation is called JAP and may be found under <https://anon.inf.tu-dresden.de/>. JAP is many ways similar to the capabilities of Tor. The network was at the time of writing a lot smaller (10 JonDos compared to 6500 relays in the Tor network).

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

12.2.11 AP3 (2004)

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

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

12.2.12 Cashmere (2005)

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

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

No client implementation could be found on the internet. 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.

12.2.13 SOR (2012)

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

12.2.14 SCION (2017)

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

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

12.3 Distributed Hash Tables

12.3.1 Tarzan (2002)

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

12.3.2 MorphMix (2002)

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

12.3.3 Salsa (2008)

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

12.4 Dining Cryptographer Based Network

12.4.1 Herbivore (2003)

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

12.4.2 Dissent (2010)

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

In an attempt to improve Dissent **wolinsky2012dissent** introduced in [wolinsky2012dissent] a modified version. This improved version mainly addresses scalability issues of the original design. Furthermore, some information leakage and scalability flaws in the original approach were addressed.

12.5 Broadcast and Multicast Networks

12.5.1 Hordes (2002)

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

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

12.6 Distributed Storage Systems

12.6.1 Freenet (2000)

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

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

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

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

12.6.2 Gnutella (2000)

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

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

12.6.3 Gnutella2 (2002)

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

12.7 Unknown (TBD)

12.7.1 Riffle (2016)

[kwon2016riffle]



12.7.2 Atom (2016)

[kwon2016atom]



12.7.3 Riposte (2015)

[corrigan2015riposte]



12.7.4 Pung (2016)

[angel2016unobservable]



12.7.5 PIR (2018)

[angel2018pir]



12.7.6 Karaoke (2018)

Karaoke [**lazar2018karaoke**] is a low latency messaging system offering an alternative to high latency systems such as Vuvuzela or Stadium. Karaoke claims to have a latency up to 10 times lower than Vuvuzela or stadium.



12.7.7 Loopix (2017)

[piotrowska2017loopix]



12.7.8 Stadium (2017)

[tvagi2017stadium]



12.7.9 Vuvuzela (2015)

Vuvuzela was presented by **van2015vuvuzela** in [**van2015vuvuzela**]. It is a scalable anonymity system offering a high throughput between millions of users. The System has a PoC implementation written in Go. An adversary knows immediately what clients do use vuvuzela. He is however unable to match up communication peers over time. The Vuvuzela client software is available under and connects to a vuvuzela network forming a centralized infrastructure. A Vuvuzela infrastructure may handle according to its authors up to 10 million users with a average bandwith cost of 3.7KB/s per user.

The MessageVortex System

*Thinking is the hardest work there is,
which is probably the reason, so few
engage in it.*

*Henry Ford, American industrialist and
founder of Ford Motor Co.*

In this part, we create a protocol called *MessageVortex*, enabling anonymous communication. Unlike most other academic attempts, we do this on the base of an adversary, which is capable of banning our technology. We, therefore, are not able to focus solely on the anonymity property. Instead, we first collect requirements for such a system in section 13. Based on these requirements, we explain our architectural concepts and decisions in section 14. We then build an outline of our protocol focusing on the protocol's main properties, without going too much into implementation details. In section 15, we describe the protocol and its key concepts in depth. We explain all aspects relevant to the academic solution without going into implementation details.

The details of the implementation, as well as their realization in infrastructure, are in part V. For operational concerns such as route building strategies, refer to part VI.

13 Requirements for an Anonymizing Protocol

In the following sections, we first define a threat model. We then elaborate on the main characteristics of the anonymizing protocol based on the threat model.

ID	Category	Short	Description
RQ1	System	Undetectable	Protocol nodes and their traffic should be undistinguishable from accepted nodes and traffic.
RQ2	System	Equal Nodes	All nodes of the system should have similar functions, capabilities, and behavior.
RQ3	System	Zero Trust	No trust should be imposed on any infrastructure unless it is the senders' or the recipients' infrastructure.
RQ4	System	Unlinkability	Message Requirement A message must not be linkable by an adversary to either a sender or a recipient.
RQ5	System	Anonymizing	A system must be able to anonymize sender and recipient at any point of the transport layer and any point within the system unless on the senders' or the recipients' node.
RQ6	System	Accounting	The system must be able to do accounting for an entity without being linked to a real identity.
RQ7	Message	Untagable	The message should be un-tagable (neither by a sender nor an involved intermediate node).
RQ8	Message	Unbugable	The message should be unbugable (neither by the sender nor by an involved intermediate node).
RQ9	Message	Unreplayable	A message or its behavior must not be replayable.
RQ10	Operational	Bootstrapping	The system must allow to bootstrap from a zero-knowledge or near-zero-knowledge point and extend the network on its own.
RQ11	Operational	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.
RQ12	Operational	Easy handleable	The system must be usable without cryptographic know-how and with popular or common tools.
RQ13	Operational	Reliable	From a user's perspective, the system must act predictably. Messages handed over to the system should reach their destination in any case.
RQ14	Operational	Transparent	From a user's perspective, the system must act predictably. He can determine the state of a message at any given point in time.
RQ15	Operational	Available	A user must have access to a working system and its software and updates.
RQ16	Operational	Identifiable sender	A recipient of a message should be able to authenticate a sender of a message beyond a simple authentication.

Table 13.1: Summary table of requirements

These requirements listed in table 13.1 are the goal we would like to achieve. We will measure up for success or failure in section 35).

13.1 Threat Model

In this section, we define in this section two adversaries. The two adversaries are an "observing adversary (mainly doing spying) and a "censoring adversary" (Actively disrupting communication). While equal in their technical capabilities, they have different executive and legislative environments. This difference in adversaries is essential as the usage of our system differs in these two environments.

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. These actors do have 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. We assume that these actors have high funding and elaborated capabilities either themselves or within reach of their 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 wish to read any information passing through portions of the Internet.
- An adversary may wish 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 location 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.

As adversaries have different capabilities and goals, we should classify them among these boundaries as well. We, therefore, split up the adversaries into the following subclasses:

- A censoring adversary
- An observing adversary

This adversary describes a powerful state-sponsored actor with very high but not unlimited powers. It serves us as a worst-case adversary.

13.1.1 Observing Adversaries

This adversary behaves like a traditional spy. He collects and classifies information while typically hiding its activities. The adversary only observes traffic and tries to extract data from the system.

Unlike the case of a censoring adversary, we imply that in most of the cases no restrictions apply for the use of anonymizing technology from a jurisdictional point of view. If restrictions apply, then such an adversary should be classified as censoring adversary, as the technology is "censored." Such classification must be done in this case, regardless of whether the adversary only tries to collect information or not.

13.1.2 Censoring Adversaries

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 necessarily cloak its activities and typically bans censorship circumventing actions as illegal.

In such environments may k -anonymity, as specified in [k-anonymous:ccs2003], not be sufficient for such an adversary. Instead, the *MessageVortex* system must hide all activities from such an adversary.

13.2 Required Properties for our Unobservable Protocol

In this section, we collect requirements for our system. We always first list a requirement and then explain why it is essential.

13.2.1 System requirements

RQ1 (Undetectable): *Protocol nodes and their traffic should be undistinguishable from accepted nodes and traffic.*

Users are unable to limit the route of our packets through named jurisdictions. Therefore, we must protect users of *MessageVortex*(users) from being subject to legal prosecution in any country. Therefore, these users 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 (mainly the routing nodes).

Threat model in section 13.1, 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 the sender.
- Identify the recipient.
- Identify other involved parties.
- Read messages passed or extract meta information.
- Disrupt or modify communication fully or partially. This may or may not include the possible identification of the traffic.

If users want to stay anonymous, they must protect our traffic from outside system influences. As we are unable to protect data from modification, we must hide the traffic of our application. In such a scenario, an adversary cannot block our traffic unless he is willing to disrupt communication entirely by disrupting communication of the transport protocol.

RQ2 (equal nodes): *All nodes of the system should have similar functions, capabilities, and behavior.*

This requirement protects all involved parties from possible legal prosecution. As we are unable to introduce our infrastructure or protocols, any categorization from outside or inside would lead to an information leak.

We 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. This means he may legally introduce nodes into our system.

To withstand an adversary outlined in section 13.1, the messages sent even within the system requires to be unidentifiable by attributes or content. "Attributes" may include any meta information including, but not limited to, frequency, timing, message size, sender, protocol, ports, or recipient. If we want to guarantee that a node is not identifiable as an endpoint of a message, all involved nodes must carry out equivalent operations. As soon as we have differences between routing nodes and endpoints, we can identify participating persons at entry or exit nodes.

Furthermore, it must be impossible from an observing adversary to identify message endpoints. All nodes must look equal from the outside in terms of traffic, as well as in terms of their offered functions and behavior.

The term "Equal nodes" does not necessarily mean that nodes must be indistinguishable. It merely means that given the functions, capabilities, and behavior of a node, no further information can be deduced.

RQ3 (zero trust): *No trust should be imposed on any infrastructure unless it is the senders' or the recipients' infrastructure.*

The requirements above protect from an adversary outside the system. Looking from the inside, an adversary may have access to much more information. An adversary will likely create nodes in an open system. As a consequence, trust in infrastructure is minimal.

In our model, we will work with suspicion 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. The 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 protect 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. If we are unable to trust these infrastructures, information may be leaked without any problems. So trusting these infrastructures is inevitable.
2. We should not trust all other infrastructure as an adversary can misuse data passed through it.

RQ4 (unlinkability): *A message must not be linkable by an adversary to either a sender or a recipient.*

We need a requirement guaranteeing the unlinkability between the sender and recipient from an adversary point of view. This prevents building social graphs and narrowing down groups of individuals.

RQ5 (anonymization): *A system must be able to anonymize sender and recipient at any point of the transport layer and any point within the system unless on the senders' or the recipients' node.*

Unobservability requires, according to [anonTerminology], that an item of interest (IOI) is undetectable from an uninvolved entity and anonymous for the involved entities. We therefore require:

As a result of the architecture of today's common networks, the anonymization of a sender or a re-

ceiver 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, even for a weak outside observer. This has been done in cypherpunk remailers (see section 12.2). 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. We have seen real-world attacks on such systems in the past, and some were successful (e.g., [penetClosure]).

[anonTerminology] defines anonymity as:

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

If we apply our threat model, we find that we require all users to be anonymous, regardless of whether a specific user is sending messages or not. Otherwise, such a user may become subject to legal prosecution.

RQ6 (accounting): *The system must be able to do accounting for an entity without being linked to a real identity.*

As a system may be flooded with messages, we need means to control the burden of processed messages. To separate message flows, we need means to control it by identity. Unlike other protocols, we have no identifier as we work based on the previous requirement anonymously. However: We will need some type of accounting to restrict single attackers in flooding.

13.2.2 Message Requirements

From the message point-of-view, we need to conserve privacy, which has been built up in the previous section.

RQ7 (untagable): *The message should be un-tagable (neither by a sender nor by an involved intermediate node).*

To protect a message from being followed or observed, a message needs to have certain properties. First, a message should not have, by design, any properties which can be observed when passing through the system. Any node should remove all parts which were under the control of the previous node.

RQ3 (zero trust) implies that a node may try to introduce such features into the message. As we cannot keep a node from doing so, we can define that such tags must be removed by the next node.

RQ8 (unbugable): *The message should be unbugable (neither by the sender nor by an involved intermediate node).*

Another way of breaking anonymity is that, instead of following a message through the system, an adversary may modify (bug) it so that the receiving or any intermediate node leaks its presence. In traditional messaging such bugging is done by introducing remotely hosted data or by introducing revocable certificate operations into the message stream and then observe the VA of a PKI for respective OCSP calls or CRL accesses. Our protocol handling must not depend on such external lookup mechanisms to ensure that bugging is not possible:

As with RQ7 (untagable), a recipient or an intermediate node may be identified by a download or lookup behavior of a recipient or any intermediate routing node.

RQ9 (unreplayable): *A message or its behavior must not be replayable.*

In a generic sense a node may also replay a message to highlight a messages property (e.g., the path or a size), which leads us to the following requirement:

13.2.3 Operational Requirements

In order to be realistically operated, our system needs to fulfill some additional requirements.

RQ10 (bootstrapping): *The system must allow to bootstrap from a zero-knowledge or near-zero-knowledge point and extend the network on its own.* Up until here, we described a system that is not centrally controlled. If not relying on broadcast domains, which is not feasible on a global scale, each node needs to know other nodes that may be contacted for routing purposes. We refer to the initial process of collecting routing nodes as bootstrapping.

RQ11 (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.*

It is quite common that weaknesses in algorithms are discovered. We may, therefore, not rely on a single algorithm. Instead, we must create a protocol supporting crypto agility, as described in [bsiPostQuantum].

RQ12 (easy handleable): *The system must be usable without cryptographic know-how and with popular or common tools.*

Academic systems are usually not known for focusing on user-friendliness. Users, on the other hand, are not known for their willingness to give up a lot of functionality or usability in trade for security. If we want that our system is used, many users need to use the system. This would lower the bar for bootstrapping and increase the size of anonymity sets. We, therefore, conclude that the system must be easy to handle for a user. Usually, this would be a decision related to a GUI or an end-user application but not to a system. However, if we want our system to be easy to handle, we need to take this as a requirement into account.

RQ13 (reliable): *From a user's perspective, the system must act in a predictable manner. Messages handed over to the system should reach their destination in any case.*

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].

RQ14 (transparent): *From a user's perspective, the system must act in a predictable manner. He is able to determine the state of a message at any given point in time.*

Transparent behavior is a prerequisite for reliability. If something is generating a behavior, but a user is unable to determine the reason for it (i.e., if a user is expecting a different behavior), he usually assumes a malfunction. Therefore, "reliable" means not only stable by its behavior. It also means diagnosable. 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]).

RQ15 (available): *A user must have access to a working system and its software and updates.*

If a user should be able to use the system, he needs access to other nodes and the required software, as well as its updates. This has to be considered even in an environment with a censoring adversary. So the system needs to be available.

Availability, in this specific context, may have two meanings, which do differ. A system 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").

RQ16 (identifiable sender): *A recipient of a message should be able to authenticate a sender of a message beyond a simple authentication.*

A messaging system offering unlinkability may offer sender anonymity from a recipient's perspective. If so, a sender should be identifiable in such a way, that a classification of senders is possible for a legit recipient, and impersonation is not achievable. It is important to understand that an identifiable sender does not necessarily mean that users can identify a sender as a specific party. It only means that two senders may be identified as the same sender.

14 Rationale

In this chapter, we set the course for our system. We explain why we built the protocol the way it is. We try to elaborate on our decisions and explain why the system is not built differently.

The system we describe is a four-layered system (transport, blending, routing, and accounting layer) in which each layer fulfills a specific duty. The transport layer is equal to an unmodified, common internet data transport protocol. The blending layer inserts and extracts our protocol messages into the transport layer. The routing layer does disassemble and reassemble the messages received and applies specially crafted operations, and the accounting layer keeps track of quotas and protects the resources of the system. The three *MessageVortex* layers (all layers except "transport") run on common internet end-user devices such as mobile phones or tablets.

14.1 System Design and Infrastructure

All anonymity systems listed in part III have in common that they do rely on dedicated servers providing an anonymity related service. Such specialized servers make operators of such servers vulnerable in an environment where a censoring adversary (as described in section 13.1) exists. Our approach should, therefore, be different. Instead of creating our own protocol, we describe a system where we use preexisting standard servers without modification. If we succeed to piggyback such a protocol invisibly, we may inherit the regular usage of this infrastructure as decoy traffic. Piggybacking and mimicking of protocols is not new. Protocols such as Skypemorph[mohajeri2012skypemorph] or pluggable transports for Tor (e.g., Meek, FTE, or OBFS4) use this technology successfully for censorship circumvention.

We will do this piggybacking in a protocol-agnostic manner. On the protocol level, this requires that we separate embedding of messages into the transport protocol from the rest of the system. To do this in such a way makes the system even harder to observe as routing graphs taking multiple protocols into account increase the complexity exponentially through their different properties.

The content of the message in the transport layer protocol is provided by the routing node and not by anyone else. This restriction is based on the fact that if we allow anyone else except the routing

node itself to control visible aspects of the transport layer message, the system could be misused for sending transport layer messages. To give an example: Such a system could be misused for blackmailing of a user not participating in the system. We simply create a message obfuscating the source and then exit the system by embedding the true blackmailing message.

As we rely on third-party infrastructure with our approach, now we have to take special care when designing our approach not to violate requirement RQ3 (zero trust). For obvious reasons, a direct connection between the sender and recipient via any named transport protocol would violate the requirement RQ4 (unlinkability). A single intermediate node would minimally imply trust in this node and its anonymization capabilities, which is not acceptable due to the requirement RQ3 (zero trust). When using multiple nodes, other anonymization protocols typically use three to five intermediate nodes due to their arguing. Typically we have at least three anonymization nodes for obvious reasons and sometimes an entry and exit node summing up to five nodes. This implies that the routing of our protocol is required. As we have an RQ3 (zero trust) policy, decisions for routing may no longer take place on the routing node but must be dictated externally.

For routing, we will use end-user devices. This decision is further backed by the requirement RQ2 (equal nodes). It, however, opposes the requirement of RQ13 (reliable), as such system participants are likely to be unreliable due to missing network connectivity, device failure due to drained batteries, or simply because they no longer participate in a network. To counter this, we implement measures on the message level.

14.2 Message and Routing

One of the biggest weaknesses of all protocols is the information leakage they have by design and the inability to restrict access to their functionality. We will build the messages with the following design guidelines:

- No routing controlled content shall survive a hop.

For us, this means that by design, a message is received and dismantled. Any content visible or manipulable by the previous node must be removed. Only new content or content inaccessible to the previous node may be used to build new messages. Following this criterion, we automatically fulfill the requirement RQ7 (untagable).

- A routing node may efficiently identify a message sender.

The sender must be efficiently identifiable. At first sight, this requirement is non-fitting as it opposes heavily to RQ5 (anonymization) and RQ4 (unlinkability). On the other hand, not providing these means makes it next to impossible to create a system that may not be misused and flooded. As the identification is pseudonymous, it must be short-lived, and multiple identities of the same sender must not be linked to each other. We will refer to this identity as an ephemeral ID (eID). This eID is handled in such a way that not complete decoding of the message is required to authorize the user. Instead, we build a message in such a way that tamper-proof, small-sized parts of the message are decoded first, and possibly bloated message content may be decoded after it is clear that the content is acceptable. If we assign "costs" to the creation of eIDs, it effectively protects the system from flooding.

- The routing operations must not leak more information than the next hop.

We will apply a transformation on each routing hop to the message. This prevents following the message throughout the system. In most of the systems, messages are mainly disassembled and reassembled or onionized. The traffic is then cloaked in mimic traffic, making it impossible for an outside observer to identify message flows. The node generating the traffic is, however, well aware of the true message flow. In our system, we will add, instead of mimicking traffic, redundancy information (or remove it). By doing so, a routing node has no longer the insight which part of the traffic is relevant to the message and which not. We may, furthermore, introduce the

possibility of distributing the message content throughout multiple paths in such a way that each path has not sufficient content to rebuild the message. In fact, depending on the complementing missing message, any content may be valid.

- Messages are protected from being replayed.

In former systems, message paths were highlighted by injecting additional information. Our system is already protected from such injections by the eID concept, which identifies the sender. There are, however, other means to highlight traffic. An adversary may either inject message payload (corrupting the message flow) or replay the message. While we cannot keep anyone from sticking to rules, we may at least implement replay protection. Furthermore, we may later observe that we are able to control willingly induced, size mismatching content.

- Messages in the routing system are "store and forward".

All synchronous routing systems have in common that message observation is relatively easy for an outside or inside observer unless mimick routes are used. This is why we allow the message to be stored and picked up or sent at a later stage.

- Use the Reed-Solomon-Function as the main routing operation.

Originally [**reed1960polynomial**] introduced a system allowing the use of polynomials to create error-correcting codes. In [**chaum1988multiparty**] **chaum1988multiparty**, they have shown that the codes are suitable for distributing data assuming enough parties are honest and not malfunctioning. Unlike **chaum1988multiparty** 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. The operation has proven to be insecure as it leaks properties of the operation applied in the result. We will show this in section 29.3. Thus, We hardened the operation to remove the negative effects.

- *MessageVortex* must provide a variety of algorithms and operations to build a message.

As all systems and algorithms applied to the system may be weakened or fail, a system needs to have the possibility to choose from multiple algorithms, protocols, and infrastructures. This choice should be made by a trustworthy system that restricts us to either the sender or the receiving system. The German Federal Office for Information Security (BSI) makes in [**bsiPostQuantum**] recommendations for systems and protocols, which we will follow.

The main text can be boiled down to the following recommendations:

- A protocol or system should be crypto agile.
- A protocol or system should use hash-based signatures for updates.
- The document furthermore recommends using symmetrical keys with a key length of 128 bit or more.
- The document recommends a combination of big long term keys and small short term keys.
- The document recommends using a combination of multiple independent algorithms in cascaded forms so that if one algorithm fails, the other one is still able to protect the data.
- For key exchange, BSI recommends lattice-based cryptography.

14.3 Summarizing Chosen Approaches for *MessageVortex*

In this section, we have taken the following decisions for *MessageVortex*:

- Piggybacks common protocols.

- Does not require specialized hardware within the Internet.
- No proprietary systems on the Internet.
- Runs on commodity hardware.
- Sends messages in an asynchronous mode.
- Creates unidentifiable decoy traffic by using a Reed-Solomon-Function.
- Have no strict message size and avoid strictly increasing or decreasing sizes in any type of message or message part.
- Does not enforce specific attributes such as transport protocol, message size, message timing, or providers.
- Run special routing operations instead of traditional mixing and recombination methods.
- Offer a choice of algorithms to use when routing.

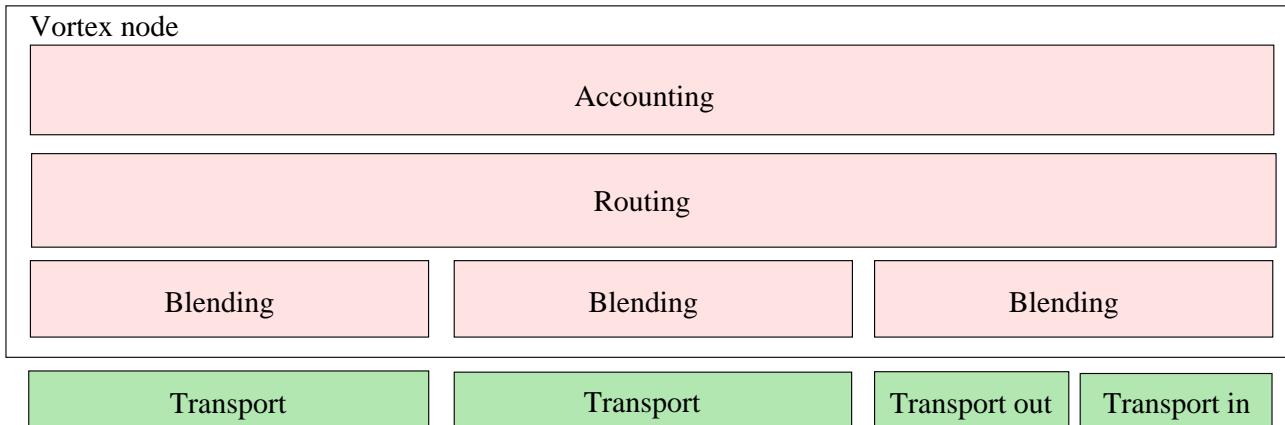


Figure 14.1: The protocol layers

The protocol is a four-layer protocol, as shown in figure 14.1. We communicate with standard protocols, which we refer to as the transport layer. While included in the message flow, they do not form a part of the *VortexNode*. The *VortexNode* itself consists out of the three layers “Blending”, “Routing”, and “Accounting”.

The blending layer is the bridging part linking a transport layer to the *VortexNode*. It injects and extracts messages from the transport layer and passes the extracted messages to the routing layer. It may be either used as a protocol bridge (e.g., in the case of XMPP) or act as a sophisticated router (e.g., in the case of the email protocols, where mails are fetched or received on push event via POP3 or IMAP while sending messages using SMTP).

The routing layer receives unified standard messages from the blending layer processes them, possibly extracts messages for local delivery, and passes subsequently created messages to the blending layer.

This design is definitely implementable on a consumer device. On the other hand, it is scalable and suitable for a clustered environment. Blending can be done in a stateless manner, is even suitable for serverless computing, and thus largely scalable. Routing may be implemented either with horizontal partitioning along with a set of eIDs or on a serverless base with a unified storage in the background. The accounting layer acts as a controller and may be implemented as well as stateless service with a minimal NOSQL-storage for all eIDs.

15 Protocol

MessageVortex is a protocol piggybacking standard transport protocols somehow similar to S/MIME[[RFC2015](#)] or PGP[[PGP](#)]. Unlike these protocols, we need the capability to keep the presence of our messages secret. The message itself should only be visible to an intended node. *MessageVortex* itself is agnostic to the transport, but we do require appropriate blending to hide within the transport protocol. The information processed on a node and its associated meta-information should not leak any information about the processed message.

Our system sends so-called *VortexMessages*. These messages are hidden within a transport protocol (e.g., SMTP or XMPP) with a blending mechanism (e.g., the steganographic algorithm F5) and extracted by a blending layer. The extracted *VortexMessage* is handed over to a routing layer. The *VortexMessage* itself contains a header block, a routing block, and possibly some payload blocks. The header block contains all the information required to protect the system. The routing block contains instructions (so-called "operations") how the payload blocks have to be processed and where to send the resulting blocks. Those operations are one of the keys as information leaking happens in this step in most of the systems. We, therefore, crafted all operations very carefully to keep as much information secret as we could. These operations are the key to the system as they allow us to increase and decrease the size of a message without leaking what part of the data is a decoy and what not.

A payload may either be kept by the system for later processing with other messages, processed (possibly with different) payload blocks, or displayed to the "local user" as a message.

The general idea of the protocol is to form a network from nodes that mix and route messages between the sender and receiver. A routing block builder (RBB), which is typically identical to the sender, has full control over almost all attributes of the message, and nodes are unable to learn anything from the message while routing. Each user has a node, and there may be additional nodes (public routing nodes) without a user connected to it.

The message is either onion-like encrypted, split into parts and remerged, or blown up with redundancy information.

This behavior results in a mixing like a system with a decoy generation in which even decoy generating nodes are unable to differentiate between real traffic and decoy as all blocks always contain parts of the message. Routing decisions are controlled by the builder of the routing block, and redundancy is possible and controlled by the routing block builder to make the system more stable.

In the following sections, we describe this protocol in detail. First, we build a terminology implicitly used in the previous chapters. Then we describe the key concepts and techniques of the protocol without in-depth analysis or reasoning. Implementation and operational aspects are discussed in part V and part VI.

15.1 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.
- **User:** Any entity, running a *MessageVortex* node.
- **Router:** Any node which is processing the message. Please note that all *VortexNodes* are routers. This includes the senders' and recipients' node.
- **Message:** The "real content" to be transferred from the sender to the recipient.

- **VortexMessage:** The encoded message passed from one node to another one. The *VortexMessage* is considered before any embedding takes place. If embedded, we refer to such a message as "embedded *VortexMessage*".
- **Payload:** Any data transported in a *VortexMessage* between routers with exception to the routing and header block, regardless of the meaningfulness or relevance to the *VortexMessage*.
- **Decoy traffic:** Any payload 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. An Identity is always assigned to a node, but one node may have multiple identities.
- **eID:** 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 recipient.

15.2 Key Components

The following sections list some of the key components of the system. Their understanding is essential for the understanding of the protocol as a whole.

We first describe a single node and its identity. This node is always equivalent to a potential recipient.

We then introduce the concept of workspaces and ephemeral identities. These concepts are essential for the routing and accounting layers. They dictate memory and storage requirements and lay a foundation for the routing layer.

A node always consists of three layers and one or more transports connected to it. The understanding of their inner workings is essential to the understanding of the project as a whole. We emphasize on their main function and their inner workings without going into implementation details. These details are further discussed in part V. We mainly focus on the data and the high-level processing done within these layers.

15.2.1 Nodes and their identities

We refer to a *VortexNode* (node) as a system run by an individual containing a processing software processing *VortexMessages*. Each node is connected to a transport layer protocol service (e.g., an IMAPv4 server as an endpoint for email or an XMPP server).

Each node o has at least one identity reflected by an asymmetric key pair K_{host_o} . Any node p communicating with node o must have the public key $K_{host_o}^1$ of the node.

A node requires the key $K_{host_o}^1$ to encrypt a message for node o . This key know-how enables environments with censoring adversaries to withstand probing attacks, as, without the knowledge of such keys, no reply from a node is received. The transport endpoint itself is not secret. The usage as *VortexNode*, however, is kept secret as long as the key is not known.

The protocol itself has the possibility to answer clear-text requests. So-called "public nodes" (see 23.3.2) make use of such messages. They are, however, an exception. In general, all *VortexMessages* are encrypted.

15.2.2 Workspaces and Ephemeral Identities

We dumped the approach for a system global unified storage for all message processing as such a design would allow an adversary to flood our storage. Instead, we introduced temporary storages suitable for a set of transaction belonging to a single identity or limited set of entities which collaborate. In our system, every transaction on a node is assigned to an ephemeral identity (eID). An eID has a limited lifetime and is represented by an asymmetric key pair and has to be created on each *VortexNode* taking part in a message processing. Each eID has a storage assigned to which we refer as “workspace”.

An eID is unique on every host and created on each *VortexNode* by the routing block builder (RBB). To create an eID, an RBB first sends a message with only a header block to the respective *VortexNode*. The request contains the new identity, a reply block, and a request to create a new identity. The receiving *VortexNode* will then typically send a challenge back. A challenge may be the start of a hash bit sequence (also referred to as “puzzle”). The requester has then to resend the request with a header block. The requester must insert additional data in such a way that the start hash in its binary form matches the bit sequence provided. Another possibility is to request a payment in cryptocurrency. This allows us to commercialize routers in some countries where the usage of such routers is generally allowed.

The length of the requested bit sequence is chosen by the accounting layer at its own will. If the request is not answered in a given time, the eID will be discarded. Analogous to an SYN-Flood attack, an adversary may try to overwhelm a *VortexNode* with eID creation requests. Such flooding will be much more costly for the adversary than for the *VortexNode*, and such a node may decide to temporarily no longer accept new eID requests without affecting already existing eIDs.

Each eID has a lifetime, a maximum number of messages to be processed, and a maximum number of bytes to be sent assigned to it. The lifetime of an eID is typically days and maybe up to a low number of months. Lifetimes may not be extended and are defined by the sender of the request. A node may accept or decline the request if the lifetime of the request or the state of the node does not meet its expectation. The puzzle sent in return may be a fixed value or related to the nodes’ current state and load.

This system guarantees that a sender must invest considerable work (in terms of CPU time required) prior to using resources of a *VortexNode*. A *VortexNode* may raise the complexity of its puzzles when having a high load. This allows for a single user to still obtain an eID while increasing costs for an attacker considerably. Even if someone floods a node with new eIDs, already created eIDs are not affected as their workspace has already been allocated.

This concept has certain downsides related to the expiry of eIDs. We will address them in section 19.3.1 and section 19.3.3.

15.2.3 Protocol Layers

As already introduced in 14.3, the protocol is built on multiple software 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. Typical examples for such layers are SMTP or XMPP servers.

2. Blending and subsequent layers (the Vortex infrastructure)

Any user of the Internet may provide these layers. Since these layers may be only Vortex routing nodes or valid endpoints, the nodes 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 the Vortex infrastructure layers into

(a) Blending layer

This layer receives messages from the Vortex system and creates transport layer conformant messages and vice-versa. In an ideal case, the messages generated by this layer are indistinguishable from any regular message traffic of the transport layer, and the embedded message is only detectable by the receiving node.

(b) Routing layer

The routing layer disassembles and reassembles messages.

(c) Accounting layer

The accounting layer has three jobs. First, he has to authorize the message processing after the decryption of the header block by the blending layer. Secondly, he handles all header request blocks and the reply blocks. And third, it keeps track of the accounting regarding the sent messages. Its main purpose is to protect the system from misuse or flooding.

In total, we have four layers. The bottom-most layer consists of unmodified standard infrastructure for transport within the Internet, and the three layers on top build a single *VortexNode*.

15.2.4 Transport Layer

The transport layer is a standard protocol within the Internet. It is neither a *MessageVortex*-specific infrastructure, nor has it been modified for the purpose. Instead, it serves the purpose as a store and forward medium. This medium solves two major problems. First, no NAT traversal technology such as "TCP hairpins" or "hole punching" is required. And secondly, it compensates for short outages due to regional routing problems to the end-user.

A transport layer should have some generic properties:

- Widely adopted
- Reliable
- Symmetrical built

For a more detailed description of the criteria, see section 16.5.1.

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 or in a broadcast-based network with a minimum amount of work for setup and deployment. The API we used may, however, be used to support almost any kind of transport protocol.

In section ??, we make a short analysis going through some common protocols outlining the strength and weaknesses of common transport protocols within the Internet.

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.

15.2.4.1 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. Another demanding task in this context is to create credible content for the transport message itself.
- 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 such as POP and IMAP require that messages are deleted from time to time to stay below the sizing quotas of an account. To manage this transport layer account is the job of the blending layer.

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. This means that some messages.

The blending is currently done by merging the *VortexMessage* using either F5 as described in [f5] or by doing plain blending, which is a binary embedding. This means that we require jpeg images included in the SMTP message.

Processing of a Message received from the Transport Layer

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

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

Processing of a Message received from 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.

Credible Content Creation for the Transport Layer

One of the most demanding tasks of the blending layer is to create transport protocol messages. In [oakland2013-parrot], **oakland2013-parrot** expresses that it is easy for a human to determine decoy traffic as the content is easily identifiable as generated content. While this is up to all that we know true, there is a possibility here to generate "human-like" data traffic to a certain extent. For the blending layer, it is not necessarily required to mimic human messages. Instead, the blending layer may generate messages such as password recovery messages, monitoring messages, and even UBE-like content. All these messages have required properties in common. First, all of them are machine-generated messages which are modified quite often. All of these messages are known to be sent and possibly adapted individually.

For the blending itself, we required a steganographic algorithm. After reviewing the options, we decided to go for F5[f5] as a steganographic algorithm. It is a reasonably well-researched algorithm, which attracted many researchers. The original F5 implementation had a detectable issue with artifacts[F5broken] caused by the recompression of the image. This issue was caused only due to a problem in the reference implementation, and the researchers have provided a corrected reference implementation without the weakness.

We looked for other steganographic algorithms, but were unable to find any other suitable algorithm apart from F5, which fulfilled the following set of criteria:

- Unbroken.
- Researched.
- Suitable for embedding in lossy compressed, common image formats (e.g., jpeg).
- An implementation or a well-specified algorithm exists.

We decided to keep our plain embedding algorithm in the implementation. It already requires an in-depth analysis or a human to detect embedding, and the message itself is, even if detected, well-protected. Its biggest strength is, however, its efficiency. This algorithm is, however, only suitable for public nodes matching up to an observing adversary (as defined in section 13.1). It must not be used in environments where a censoring adversary is suspected.

When using F5, jpeg images are required. Imagery requires to be at least eight times the size of the message embedded. Unlike other approaches harvesting random pics or obtaining them from a local repository, we recommend using machine-generated images such as rendered content. We recognize that custom Gravatars, router and usage graphs of services, or render services are suitable imagery material for our purpose. The message content would obviously be machine-generated content and not being suspect. This would effectively render the dead parrot problem as described in [oakland2013-parrot] ineffective.

15.2.4.2 Routing Layer

A routing layer needs to receive all payload and routing blocks, and process them (For an exact outline of the routing block, see section 15.2.5). These blocks are stored in a suitable list within the workspace of the eID identified by the header block.

Within the routing block, we find a set of instructions, next *VortexNodes* information, and the encrypted routing blocks for the messages to be assembled. A simplified representation of a routing block is shown in figure 15.1.

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 defined in the header block. Anytime during that time window, a routing layer processes the routing instructions contained in the assembly operations

$$\text{ROUTING}_o = \langle [\text{ROUTINGCOMBO}]^*, \text{replyBlock}, \text{operation}^* \rangle \quad (15.1)$$

$$\begin{aligned} \text{ROUTINGCOMBO} &= \langle \text{processInterval}, K_{\text{peer}_{o+1}}, \text{recipient}, \text{nextMP}, \text{nextHP}, \\ &\quad \text{nextHEADER}, \text{nextROUTING}, \text{assemblyInstructions} \rangle \end{aligned} \quad (15.2)$$

$$\text{PL} = \langle \text{payload octets} \rangle^* \quad (15.3)$$

$$\text{nextMP} = E^{K_{\text{host}_{o+1}}^1}(K_{\text{peer}_{o+1}}) \quad (15.4)$$

$$\text{nextHP} = E^{K_{\text{host}_{o+1}}^1}(K_{\text{sender}_{o+1}}) \quad (15.5)$$

$$\text{nextHEADER} = E^{K_{\text{sender}_o}}(\text{HEADER}_{o+1}) \quad (15.6)$$

$$\text{nextROUTING} = E^{K_{\text{sender}_o}}(\text{ROUTING}_{o+1}) \quad (15.7)$$

$$\text{operations} = \langle \text{list of operations} \rangle \quad (15.8)$$

$$\text{assemblyInstructions} = \langle \text{blendingInformation}, \text{nextHop}, \langle \text{mapping operation}+ \rangle \rangle \quad (15.9)$$

Figure 15.1: Simplified representation of a routing block

of the routing block. If successful, the message will be sent using the specified blending layer and target address.

The routing layer stores the main information assigned to the operation of routing messages. The following data has to be kept for routing within the eIDs workspace:

- **Build[]** $\langle \text{expiry}, \text{buildOperation} \rangle$

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.

The *buildOperation* is extracted by enumerating *operation** while *expiry* is the upper bound of the *processInterval*.

- **Payload[]** $\langle \text{expiry}, \text{payload}, \text{id} \rangle$

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.

- **Route[]** $\langle \text{processInterval}, \text{blendingInformation}, \text{nextHop}, \text{nextMP}, \text{nextHP}, \text{nextHeader}, \text{nextRouting}, K_{\text{peer}_{o+1}}, \text{assemblyInstructions} \rangle$

The list of routing information triggers processing. At a randomly chosen time defined in the *processInterval*, a message is composed. The message is assembled by doing $\langle \text{nextMP}, E_{K_{\text{peer}_{o+1}}} \langle \text{nextHP}, \text{nextHEADER}, \text{nextROUTING}, \text{payload}^* \rangle \rangle$. The payloads are created with the help of the arrays *build[]* and *payload[]*, and as soon as the message is authorized by accounting and passed to the blending layer, the entry in this list is discarded.

15.2.4.3 Accounting Layer

The accounting layer keeps tracks of all information required assigned to ephemeral identities (eID). It is queried by the blending Layer and routing Layer for authorization of the operations. The accounting layer manages the following tuples of information:

- **eID** $\langle \text{expiry}, \text{pubKey}, \text{msgsLeft}, \text{bytesLeft}, \text{temporary} \rangle$

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.

- **Puzz[]** $\langle \text{expiry}, \text{request}, \text{puzzle} \rangle$

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.

- **Replay[]** $\langle \text{expiry}, \text{serial}, \text{numberOfUsages} \rangle$

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.

15.2.5 VortexMessages

A *VortexMessage* is built by combining multiple loosely interconnected blocks. We first name the blocks and their function, and then we explain the inner working of the blocks and do some reasoning why the block has been built as it is.

Figure 15.2 shows an outline of the block structure of a message destined to $host_o$. For a mathematical representation, see figure 15.3.

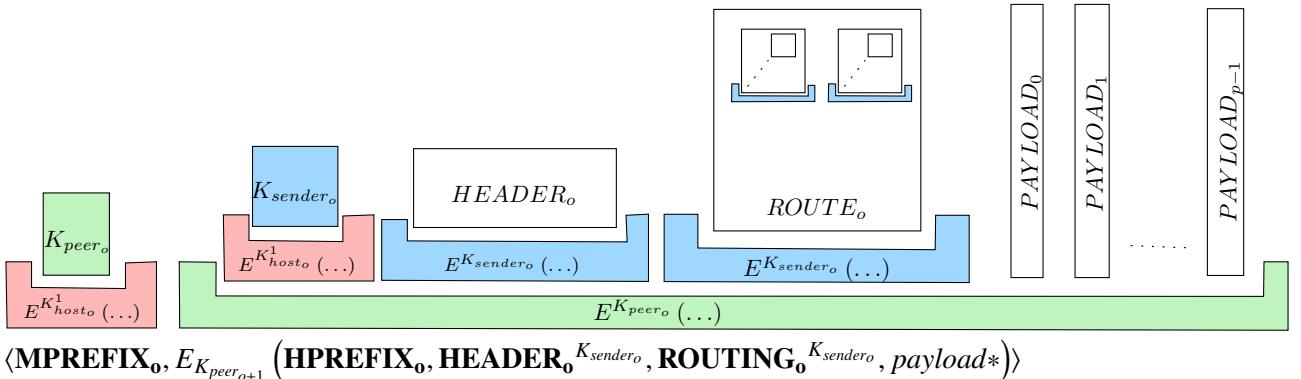


Figure 15.2: Simplified message outline visually and in math

The first block is the message prefix block **MPREFIX** _{o} , which has been encrypted with the public key of the receiving node K_{hosto}^1 . This block contains the key for decrypting the whole rest of the message. Each PREFIX block contains a symmetrical ey and the specification on how to encrypt or decrypt with it (mode, padding, IV, and other possibly required parameters) in ASN.1 encoding.

Immediately following the message prefix block, we have the inner message block. This message blocks contain three more blocks and a variable number of payload blocks. The inner message is encrypted with the symmetrical peer key K_{peer_o} . This peer key is specific to this message and nowhere reused. It is only known by the two peer hosts $host_o$ and $host_{o-1}$, and the routing block builder (RBB). More importantly, $host_{o-1}$ does not need to know the host key of $host_o$. Therefore, relaying a message to $host_o$ does not enable $host_{o-1}$ to communicate with $host_o$.

The blocks **HEADER** _{o} and **ROUTING** _{o} are protected with an additional key $K_{sendero}$. The decryption key is obtained by $host_o$ from the header prefix block **HPREFIX**. After only decrypting the header block **HEADER** and verifying its signature, the accounting layer may check if further processing is authorized. The splitting of the two keys allows us to...

- ... send a message to $host_o$ without $host_{o-1}$ knowing the host key of $host_o$.
- ... hide the structure of the message itself.

- ... keep the content of **HEADER₀**, and **ROUTING₀** secret from $host_{o-1}$.

After authorization by the accounting layer, the header block is processed as outlined in section 19.2.1. Basically, we just add the routing blocks and payload to the respective workspace and wait for the routing layer to process the information.

Looking at a full *VortexMessage*, we get the protocol outline, as shown in (15.10) on page 63.

$$\text{VORTEXMESSAGE} = \langle \text{MP}^{K_{host_o}}, \text{INNERMESSAGE} \rangle \quad (15.10)$$

$$\text{INNERMESSAGE} = \langle \text{CP}^{K_{host_o}^{-1}}, \text{H}^{K_{sender_o}}, E^{K_{sender_o}^{-1}}(H(\text{HEADER})), [\text{R}^{K_{senderN}}], [\text{PL}] * \rangle^{K_{peerN}} \quad (15.11)$$

$$\text{MP}^{K_{hostN}^{-1}} = E^{K_{hostN}^{-1}}(\text{PREFIX}(K_{peerN})) \quad (15.12)$$

$$\text{HP}^{K_{hostN}^{-1}} = E^{K_{hostN}^{-1}}(\text{HPREFIX}(K_{senderN})) \quad (15.13)$$

$$\text{H}^{K_{senderN}} = E^{K_{senderN}}(\text{HEADER}) \quad (15.14)$$

$$\text{HEADER} = \langle K_{senderN}^1, \text{serial}, \text{maxReplays}, \text{validity}, [\text{requests}, \text{requestRoutingBlock}], [\text{puzzleIdentifier}, \text{proofOfWork}] \rangle \quad (15.15)$$

$$\text{R}^{K_{senderN}} = E^{K_{senderN}}(\text{ROUTING}) \quad (15.16)$$

$$\text{ROUTING} = \langle [\text{ROUTINGCOMBO}]*, \text{forwardSecret}, \text{replyBlock}, \text{operations} \rangle \quad (15.17)$$

$$\text{ROUTINGCOMBO} = \langle \text{processInterval}, K_{peerN+1}, \text{recipient}, \text{nextMP}, \text{nextHP}, \text{nextHEADER}, \text{nextROUTING}, \text{assemblyInstructions} \rangle \quad (15.18)$$

$$\text{nextMP} = E^{K_{host_{o+1}}^1}(K_{peer_{o+1}}) \quad (15.19)$$

$$\text{nextHP} = E^{K_{host_{o+1}}^1}(K_{sender_{o+1}}) \quad (15.20)$$

$$\text{nextHEADER} = E^{K_{sender_o}}(\text{HEADER}_{o+1}) \quad (15.21)$$

$$\text{nextROUTING} = E^{K_{sender_o}}(\text{ROUTING}_{o+1}) \quad (15.22)$$

$$\text{operations} = \langle \text{list of operations} \rangle \quad (15.23)$$

$$\text{assemblyInstructions} = \langle \text{blendingInformation}, \text{nextHop}, \langle \text{list of mapping operations} \rangle \rangle \quad (15.24)$$

$$\text{PL} = \langle \text{payload octets} \rangle * \quad (15.25)$$

$$(15.26)$$

Figure 15.3: Detailed representation of a VortexMessage

The routing log block is an onionized block. It contains at least a *forwardSecret*, which must match up with the header blocks *forwardSecret*. This mechanism is required to guarantee that routing blocks are not exchanged. The *replyBlock* provides a possibility, if provided, to contact the original sender of the message without knowing him. It is just a routing block with instructions on how to prepare the message to be sent. The routing combos contain all the necessary information and prebuilt blocks to create the subsequent messages.

At the very end, we got the payload. These blocks are simply added to the eIDs workspace.

The double encryption of the routing and header block, are doubly encrypted. We could argue that the inner message block should not be encrypted with a peer key. This looks like a flaw at first sight but is, in fact, a feature that is very important. 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* is 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.

15.2.5.1 Message Structure Related to Censorship Circumvention

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.

The whole *VortexMessage* is, looking from outside, a structureless blob with a maximum of entropy caused by the encryption employed.

This is intentional and by design. Plain embedding uses furthermore a method of splitting, which allows a message block to be embedded in chunks in carrier information. By design, neither the message nor their embedding display detectable attributes allowing them to identify the message.

Exactly as with the routing operations, great care has been applied. Any random sequence of bytes may be interpreted as valid chunking. For more exact implementation details on chunking, see section 18.1.1.1.

15.2.5.2 Message Structure Related to Information Leaking

From the inside, the **INNERMESSAGE** (see 15.11) is built as a structure leaking the absolute minimum of information. A node receiving and decoding the message will learn the following information:

- IP of the sender of the transport layer.
- The address and embedding schemes of all receiving transport layers.
- The size of the payload blocks.
- The size of the subsequent routing blocks.
- The peer key K_{peero} .
- The size of the prefix blocks.

It is unable to extract the following information:

- The required keys to communicate with the suspected peer node.
- Any information related to message size, content, or recipient.

15.2.6 Routing Operations

The routing operations build the core as they define the capabilities of the mixing. We decided to introduce three different classes of operations. Wherever we employ crypto, operations, we may choose the operation required for the operation. No choices exist for the core Reed-Solomon-Function, the padding and spitting operation related to it, and the split and merge operations.

15.2.6.1 *addRedundancy* and *removeRedundancy* Operations

These operations build the core of the routing capabilities of a node. The operation allows an 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 15.4.

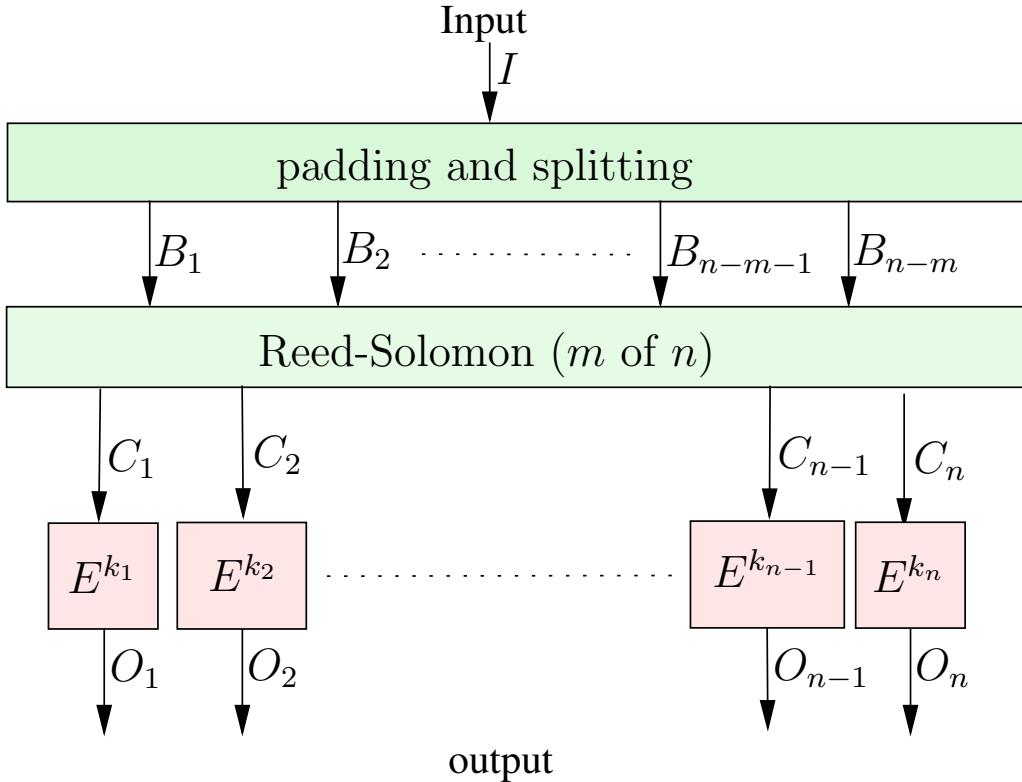


Figure 15.4: Outline of the *addRedundancy* operation

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.

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 $RS_{m \text{ of } n}$ (allows adding m redundancy blocks) and an encryption block \mathbf{M} sized K as follows:

$$i = \text{len}(\mathbf{M}) \quad (15.27)$$

$$e = k \cdot n \quad (15.28)$$

$$l = \left\lceil \frac{i + 4 + C2}{e} \right\rceil \cdot e \quad (15.29)$$

$$p = i + \left(C1 \cdot l \pmod{\left\lfloor \frac{2^{32} - i}{l} \right\rfloor \cdot l} \right) \quad (15.30)$$

$$\mathbf{X} = \langle p, \mathbf{M}, R_t(s, l - i - 4) \rangle \quad (15.31)$$

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

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 $\text{len}(\mathbf{X}) < \text{size} < (\text{len}(\mathbf{X}) - k \cdot n)$ is no longer true and any value smaller $\text{len}(\mathbf{X}) - k \cdot n$ may be correct as well.
3. An RBB may predict the exact binary image of the padded message when specifying $C1$, $C2$, and $R_t(s,)$.
4. A node knowing the original parameters $C1$, $C2$, and the initial PRNG seed s can detect successful decryption.

Apart from being non-standard padding, the padding has additional downsides:

- The padding is inefficient compared to simple paddings such as PKCS#7
- The padding requires an initialized PRNG to generate the padding.
- Depending on the chosen parameters, the padding overhead may become significant.

After the padding, the date is ready for the Reed-Solomon part of the 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 later.

$$t = n - 1 \quad (15.32)$$

$$\text{inbytesA} = \text{vec2mat}\left(\mathbf{X}, \frac{\text{len}(\mathbf{X})}{m}\right) \quad (15.33)$$

$$\mathbf{V} = \begin{pmatrix} 0^0 & 0^1 & 0^2 & \dots & 0^{(m-1)} \\ 1^0 & 1^1 & 1^2 & \dots & 1^{(m-1)} \\ 2^0 & 2^1 & 2^2 & \dots & 2^{(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t^0 & t^1 & t^2 & \dots & t^{(m-1)} \end{pmatrix} \quad (15.34)$$

$$\mathbf{P} = \mathbf{VA}(GF(2^\omega)) \quad (15.35)$$

$$\langle \mathbf{Q}_1, \dots, \mathbf{Q}_n \rangle = \text{row2vec}(P) \quad (15.36)$$

$$R_i = E^{K_i}(Q_i) \quad (15.37)$$

We apply the Reed-Solomon function by employing a Vandermonde matrix (\mathbf{V}). We build the data matrix (\mathbf{A}) by distributing the data into $\frac{\text{len}(\mathbf{X})}{m}$ 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 29.1 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 ciphertext. We observed faint but statistically relevant reminders of the unencrypted graphs when treating the same block with the same key and different redundancy parameters.

15.2.6.2 *encrypt and decrypt* Operations

The encrypt and decrypt operations are essential for the requirement that tagging should not be possible. Unlike the *addRedundancy* and *removeRedundancy*, the splitting operations do not feature any encryption step after splitting or merging. Reusing a payload block that has only been split or merged would repeat the payload pattern on multiple nodes during transfer. That is why we require to have encryption.

The reason for not building this step into the split and merge function was simple. We needed a separate encryption step to be able to work as an onionizing system, and there were use cases where integrated encryption did not make sense. For further details on this topic, see section ??.

15.2.6.3 *mergePayload and 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 $\text{len}(pb_1)$ 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:

$$split(f, pb_1) = \langle pb_1, pb_2 \rangle \quad (15.38)$$

$$pb_1.startsWith(pb_2) \quad (15.39)$$

$$pb_1.endsWith(pb_3) \quad (15.40)$$

$$len(pb_2) = floor(len(pb_1) \cdot f) \quad (15.41)$$

$$len(pb_1) = len(pb_2) + len(pb_3) \quad (15.42)$$

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 Merge-Payload Operation pb_3 have to follow the following rules:

$$merge(pb_1, pb_2) = pb_3 \quad (15.43)$$

$$pb_3.startsWith(pb_1) \quad (15.44)$$

$$pb_3.endsWith(pb_2) \quad (15.45)$$

$$len(pb_3) = len(pb_1) + len(pb_2) \quad (15.46)$$

Unlike other operations, this operation has no encryption step attached to it. We usually attached an encryption step to remove repeating patterns from the *VortexMessagestream*.

It has to be mentioned that this operation tuple has some issues when it comes to floating-point implementations. They are solvable but had to be specified unexpectedly precisely in order to enable a true cross-platform implementation. For more information regarding the issue and exact implementation, see section 19.2.3.

15.3 Summary

The *MessageVortex*-Protocol is split into the four layers "Transport" (a common internet standard protocol), "blending" (extracting and embedding *VortexMessages*), "Routing" (A layer reassembling messages according to received instruction), and "Accounting" (Keeps track of all stored data and discards expired information).

All nodes are realized in decentral devices such as computers or mobile phones. Messages are hidden with either plain embedding or (referred) F5 in the transport layer message. The routing layer processes messages by applying operations to messages. Valid operations are encrypt or decrypt a message chunk, split a message chunk into two parts, merge two parts into one, or add or remove redundancy information. The last operation is the most valuable. This operation allows by employing an extended Reed-Solomon-Operation to add decoy traffic to the message flow without enabling a node to identify such traffic. Furthermore, it allows a sender to send parts of a message through multiple chains of routing nodes to a recipient. Each message itself does not leak the message content since depending on the completing block, any message with appropriate length may be valid.

The routing itself is done in a temporarily allocated storage called "workspace", which is tied to an ephemeral identity (eID) represented by an asymmetric key pair. To get an eID, a sender typically solves a crypto puzzle.

Implementation

No matter how hard you work, someone else is working harder.

Elon Musk, entrepreneur

The implementation differs from the academic model in some details. It is foremost more precise than the academic model. Furthermore, it requires a strict definition of the implementation in order to guarantee the interoperability between different implementations.

This section focuses therefore on the details of the reference implementation.

16 Selection of Algorithms, Encodings, and Protocols

In this chapter, we choose the following mandatory supported algorithms:

- Encoding: ASN.1

- Encryption
 - AES128/256
 - Camellia128/256

- Modes
 - ECB
 - GCM

- Paddings
 - PKCS#1
 - PKCS#7

- MACs
 - SHA256/512
 - RIPE-MD256

- PRNG
 - mrg32k3a
 - blumMicali

Where security relevant we always choose two independent algorithms. As our protocol has the means of signaling them, we may support additional algorithms without affecting communication while improving the variety of available algorithms.

In the following sections, we emphasize on the choice and the encoding used on the protocol level.

For all algorithms we apply the following criteria:

- Always focus on common standards
- Focus on interoperability when selecting standards
- Focus on efficiency (wherever possible use simple, parallelizable algorithms)
- When sensible and possible chose at least two unrelated algorithms (e.g., cryptographic algorithms or MACs)



16.1 Encoding Scheme

As encoding scheme we specified ASN.1[dis19878824]. It is more compact than the originally selected XML-Standard. It is very common in the fields of telecommunication and encryption (e.g. X509 certificates are represented in ASN.1). To maintain biggestmost interoperability, we choose DER encoding as it has exactly one possible representation for every value. This is especially important when doing signing or solving puzzles in our case.

On the downside ASN-1 encoding is, unlike XML, not human readable. As we hide the messages anyway, we considered this as minor flaw, as we need to have an extracting program anyway to see the content of a message.

16.2 Cipher Selection

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

We decided to define fixed key sizes for symmetric ciphers as we went with block ciphers. For the asymmetric ciphers we encode the keysize in the parameters as they are due to their differences far more often flexible.

```

1  CipherSpec ::= SEQUENCE {
2    asymmetric [16001] AsymmetricAlgorithmSpec OPTIONAL,
3    symmetric   [16002] SymmetricAlgorithmSpec OPTIONAL,
4    mac         [16003] MacAlgorithmSpec OPTIONAL,
5    cipherUsage [16004] CipherUsage
6  }
7
8  CipherUsage ::= ENUMERATED {
9    sign        (200),
10   encrypt     (210)
11 }
12
13 SymmetricAlgorithmSpec ::= SEQUENCE {
14   algorithm   [16101]SymmetricAlgorithm ,
15   -- if omitted: pkcs7
16   padding     [16102]CipherPadding OPTIONAL,
17   -- if omitted: cbc
18   mode        [16103]CipherMode OPTIONAL,
19   parameter   [16104]AlgorithmParameters OPTIONAL
20 }
21
22 AsymmetricAlgorithmSpec ::= SEQUENCE {
23   algorithm   AsymmetricAlgorithm ,
24   -- if omitted: pkcs1
25   padding     [16102]CipherPadding OPTIONAL,
26   parameter   AlgorithmParameters OPTIONAL
27 }
28
29 SymmetricKey ::= SEQUENCE {
30   keyType     SymmetricAlgorithm ,
31   parameter   AlgorithmParameters ,
32   key         OCTET STRING (SIZE(16..512))
33 }
34
35 AsymmetricKey ::= SEQUENCE {
36   keyType     AsymmetricAlgorithm ,
37   -- private key encoded as PKCS#8/PrivateKeyInfo
38   publicKey   [2] OCTET STRING,
39   -- private key encoded as X.509/SubjectPublicKeyInfo
40   privateKey  [3] OCTET STRING OPTIONAL
41 }
42
43 SymmetricKey ::= SEQUENCE {
44   keyType     SymmetricAlgorithm ,
45   parameter   AlgorithmParameters ,
46   key         OCTET STRING (SIZE(16..512))
47 }

48
49 AsymmetricKey ::= SEQUENCE {
50   keyType     AsymmetricAlgorithm ,
51   -- private key encoded as PKCS#8/PrivateKeyInfo
52   publicKey   [2] OCTET STRING,
53   -- private key encoded as X.509/SubjectPublicKeyInfo
54   privateKey  [3] OCTET STRING OPTIONAL
55 }
56
57 SymmetricAlgorithm ::= ENUMERATED {
58   aes128      (1000), -- required
59   aes192      (1001), -- optional support
60   aes256      (1002), -- required
61   camellia128 (1100), -- required
62   camellia192 (1101), -- optional support
63   camellia256 (1102), -- required
64   twofish128  (1200), -- optional support
65   twofish192  (1201), -- optional support
66   twofish256  (1202) -- optional support
67 }
68
69 AsymmetricAlgorithm ::= ENUMERATED {
70   rsa          (2000),
71   dsa          (2100),
72   ec           (2200),
73   ntru         (2300)
74 }
75 ECCurveType ::= ENUMERATED{
76   secp384r1   (2500),
77   sect409k1   (2501),
78   secp521r1   (2502)
79 }
80 AlgorithmParameters ::= SEQUENCE {
81   keySize     [9000] INTEGER (0..65535) OPTIONAL,
82   curveType   [9001] ECCurveType OPTIONAL,
83   iv          [9002] OCTET STRING OPTIONAL,
84   nonce       [9003] OCTET STRING OPTIONAL,
85   mode        [9004] CipherMode OPTIONAL,
86   padding     [9005] CipherPadding OPTIONAL,
87   n           [9010] INTEGER OPTIONAL,
88   p           [9011] INTEGER OPTIONAL,
89   q           [9012] INTEGER OPTIONAL,
90   k           [9013] INTEGER OPTIONAL,
91   t           [9014] INTEGER OPTIONAL
92 }
93
94 CipherMode ::= ENUMERATED {
95   -- ECB is a really bad choice. Do not use unless really

```

Figure 16.1: Definition of the structures related to ciphers

From the requirements side, we have to follow the following principle: 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 algorithms 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 Camellia192 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.

16.3 Mode Selections

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 before, was on the question of whether there is an open implementation available in java, which is reasonably tested.

```
1 CipherMode ::= ENUMERATED {
2   -- ECB is a really bad choice. Do not use unless really
3   -- necessary and you are sure that the content is already
4   -- encrypted
5   ecb      (10000), -- optional support
6   cbc      (10001), -- required
7   eax      (10002), -- optional support
8   ctr      (10003), -- required
9   ccm      (10004), -- optional support
10  gcm     (10005), -- optional support
11  ocb      (10006), -- optional support
12  ofb      (10007), -- optional support
13  none    (10100)  -- required
14 }
```

Figure 16.2: Enumeration definition of modes in ASN.1 with support requirements.

Figure 16.2 shows the selected paddings and their requirement level.

Very important was, that we quite often reencrypt already encrypted content. As a result we had not to exclude algorithms such as ECB.

- 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: $H_{GFmult}(o \oplus (len(A)||len(B))) \oplus E^{K^0}(counter_0)$. 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 ECB 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.

We decided to go with mainly cbc. However, most of the implementations are available and lightweight, we therefore were not as restrictive as usual when defining a minimal set

16.4 Padding selection

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.

We have chosen for the paddings outlined in figure 16.3 to be supported.

```
1 CipherPadding ::= ENUMERATED {
2   none          (10200), -- required
3   pkcs1         (10201), -- required
4   rsaesOaep    (10202), -- optional support
5   oaepSha256Mgf1 (10202), -- optional support
6   pkcs7         (10301), -- required
7   ap            (10221)  -- required
8 }
```

Figure 16.3: Enumeration definition of paddings in ASN.1 with support requirements.

16.4.1 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 padding may give a clue if decryption was successful or not. RSAES-OAEP ist the newer of the two padding standards

16.4.2 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.

16.4.3 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 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.

16.4.4 Honorable Mention: AP Padding for *redundancy* Operations

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.

This padding is the only padding for the *addRedundancy* and *removeRedundancy* operations.

16.4.5 Pseudo Random Number Generator Selection

For our *addRedundancy* and *removeRedundancy* operations we needed a pseudo random number generator (PRNG). For our implementation we did not research deeply this part as it seemed irrelevant. The only criterion was, that it had to create content indistinguishable from an encrypted message. This criterion arose as we use it for padding invisibly an already encrypted message.

The PRNG used for our implementation is an xorshift+ generator. It is based on the XSadd PRNG [**marsaglia2003xorshift**] and passes the bigcrush PRNG test suite. It is a fast, xor based PRNG which has two internal 64 bit seed states s_0 respectively s_1 and is defined as follows:

$$x = s_0 \tag{16.1}$$

$$s_0 = s_1 \tag{16.2}$$

$$x = x \oplus (x \ll 23) \tag{16.3}$$

$$s_1 = x \oplus s_1 \oplus (x \gg 17) \oplus (s_1 \gg 26) \tag{16.4}$$

$$nextNumber = s_1 + s_0 \tag{16.5}$$

We have chosen this comparably weak PRNG for practical reasons. It is fast, simple, and is based on operations easy to implement on hardware. As we do not need a cryptographically strong PRNG, it is our primary choice so far.

As the protocol is heavily dependent on security we have introduced everywhere at least one alternate algorithm which may be used if one of the choices may become a problem. In order to have a second choice for the PRNG we define the Blum-Micali PRNG as described in [**blum1984generate**]. This PRNG is a cryptographically secure PRNG and is defined as follows:

p is prime and g is a primitive root modulo p . x_0 reflects the seed state.

$$x_{i+1} = g^{x_i} \mod p \tag{16.6}$$

16.5 Transport Layer Protocol Selection

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

We will identify SMTP and XMPP as good transport layer protocols for the *MessageVortex* approach, as they have all required properties.

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 ???. 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 ???. The list here does not reflect the quality or maturity of the protocols. It is a simple analysis of suiting as a transport layer.

16.5.1 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.

16.5.2 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 weakness is founded in the fact that traffic in this report is classified among two criteria: Known server or known port. As streaming services consume a considerable part of the Internet bandwidth (according to the report more than 60% download). The focus on the report lies on the bandwidth intense figures. However, leaving 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 according to [**kim2010today**])

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

16.5.3 Analysis

16.5.3.1 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.

16.5.3.2 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 behavior 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).

16.5.3.3 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.

16.5.3.4 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).

16.5.3.5 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).

16.5.3.6 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.

16.5.3.7 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.

16.5.3.8 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 similarly 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.

16.5.3.9 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 ?? 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.

16.5.3.10 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.

16.5.3.11 MMS

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

16.5.4 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 protocols 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 16.1 on page ?? 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	✗	✗	✗
MOTT	~	✓	✗
AMQP	~	✓	✗
CoAP	~	~	✗
WAMP	✗	✓	~
XMPP	✓	✓	✓
SMTP	✓	✓	✓

Table 16.1: 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.

17 Transport Layer Implementation

17.1 Implementation of a Dummy Transport Layer

For better diagnosability and fast setup, we implemented a custom transport layer working on a config-less manner in a localhost or broadcast-domain environment. The transport layer is based on the Hazelcast distributed hashmap. Implementation may be found under `net.messagevortex.transport.dummy.DummyTransportTrx`.

17.2 Implementation of an Email Transport Layer

Email supports a conglomerate of protocols. Looking at the client side, we will find that email is sent with an authenticated SMTP connection. The SMTP connection is somewhat different than the connections used to send emails to the destination. First of all the client port was shifted in the past to a specific submission port (SMTPTS: Port 465; Submission: Port 587). These submission ports are authenticated (either by username and password, by IP or by certificates) and usually privileged (no UBE checks). On the retrieval side, SMTP is not capable of handling these tasks sensibly. Instead, POP3 and IMAPv4 are used. POP3 is a deposit box for email where a device fetches the mail and stores it locally. This is commonly in use for automated processing of mails but these days, where the same user owns multiple devices no longer adequate. IMAPv4 offers to organize mails on the server. This allows a user to have the same folder structure of mails in a synchronized manner on all devices.

For an ideal implementation we would do the following: Organize our *MessageVortex* mails in a separate account. The account is accessed through a local proxy relaying our “ordinary outgoing mails” through the SMTP server of our regular provider and all *MessageVortex* related traffic through the provider of our *MessageVortex* mailbox. Keeping the two mailboxes separate is sensible and important as we will see in part VI. The housekeeping on the account used for *MessageVortex* is done automatically and in a sensible way, comparable to a human (e.g., handle draft, sent and trash bin folders sensibly and keep all mails in a flat structure deleting old mails from time to time). The mails from the regular and the *MessageVortex* account are merged by the proxy in a transparent way. Keeping the messages apart but offering a unified look.

In our current implementation the house keeping part has been skipped. Instead, we are just fetching the newly arrived messages and put in a local storage. The email presented to the client is provided by a local IMAP server. Persistence of these messages is not yet implemented.

17.3 Implementation of an XMPP Transport Layer

The XMPP protocol (formerly called Jabber, as specified in [RFC6120], is natively not capable of transferring anything else but text messages. Unlike email, XMPP is capable of true end-to-end signing and object encryption without solving the problem of the initial trust. While we may use the end-to-end encryption for additional security, relying on this feature is not sensible as we would put trust into the security features of an intermediate node. This would effectively violate RQ3 (zero trust) requirement. We decided to use the extension defined in [xep0231] to transfer our messages, as it is simple and reliable.

To transfer a *VortexMessage*, we could embedd a MIME message just as with SMTP. While this would be technically feasible, the usage of MIME is not common and even discouraged. Instead, the inner structure if an XMPP message relies on XML.

XMPP has an improvement process based on XEPs. For including binary contents such as attachments in messages multiple XEPs exists. Table ?? shows all idenified candidates.

Name	Status (as of 06-2020)	Purpose
xep0047[xep0047]	Final Standard	Allows sending chunked, base64 encoded data within the jabber connections.
xep0066[xep0066]	Draft Standard	Allows sending URIs of remotely hosted binary data.
xep0096[xep0096]	Deprecated (referred to XEP-0234)	Improvement of [xep0066] allowing to send metadata and alternative URIs
xep0135[xep0135]	Deferred (inactive)	Inband or Out-of-band file discovery and referral service. May be used in conjunction with FTP, HTTP, SCP, or [xep0096].
xep0231[xep0231]	Draft Standard	Allows sending inband small unchunked files and referring within the message similarly to [rfc2397].
xep0234[xep0234]	Deferred (inactive)	Based on [xep0166] allowing out-of-band content negotiation of complex data streams

Table 17.1: Overview of XEPs related to transporting binary data.

Many documents have either reached standard or have been deferred due to inactivity. We used “[xep0231](#)”[[xep0231](#)] for our protocol. It is simple to implement as a transport layer, used in many clients (e.g., Prosody, Pigdin, or CoyIM). A node may use any XMPP server regardless of any additional support for transfer.

The blending layer may generate decoy message analogue to the messages generated in the case of email.

17.4 Distributed Configuration and Runtime Store of processing content

If we care to run our *MessageVortex* nodes in a sensible and

18 Blending Layer Implementation

18.1 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 in EBNF as follows:

```
1 plainEmbedding = "("plain:<#BytesOffset>[,<#BytesOffset>]*")  
2 F5Embedding   = "(F5:<passwordString>[,<PasswordString>]*")"
```



18.1.1 Plain Embedding

The *plainEmbedding* is a simple embedding replacing parts of the original file with the content of the *VortexMessage*. To maintain the header information of a file we introduced an offset as a set of fixed values. A plain embedding may be identified by a host when decoding a single block with $K_{host_o}^{-1}$ if the resulting structure is an ASN.1 stucture is equivalent to XXX.

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 employing plain embedding is thus not suitable for production use as the messages remain identifiable or at least suspicious.

18.1.1.1 Chunking of Plain Embedded Messages

In this section we describe the chunked embedding into plain messages. The chunking is done by embedding one to four byte integer values which reflect (modulo the the remaining file size) the next chunk and the following gap. The byte layout is chosen in such a way that any byte sequence from two to eight bytes form a valid chunk header.

When plain embedding messages we have the problem that most of the files have recurring logical structures. Such structures should not be broken when embedding in plain as such embedding as it would leak in an easy detectable manner the presence of a broken file.

18.1.2 Implementation of F5 Blending



18.2 Blending Content Generation



19 Routing Layer Implementation



19.1 ASN.1 encoding scheme

Originally, we implemented the protocol as XML encoded messages. This encoding had, however, several flaws. First the huge amount of encrypted data within the document made the messages bulky and at the same time loose one of its main strengths: readability for humans. The encoding required for binary data caused messages to increase in size due to their onionized structure.

Furthermore, the complex XML features



19.2 Processing of messages



19.2.1 Processing of Incoming Messages

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 19.1. 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.



19.2.2 Processing of Outgoing Messages



19.2.3 Implementation of Operations



19.3 Request handling



19.3.1 Requesting a new Ephemeral Identity



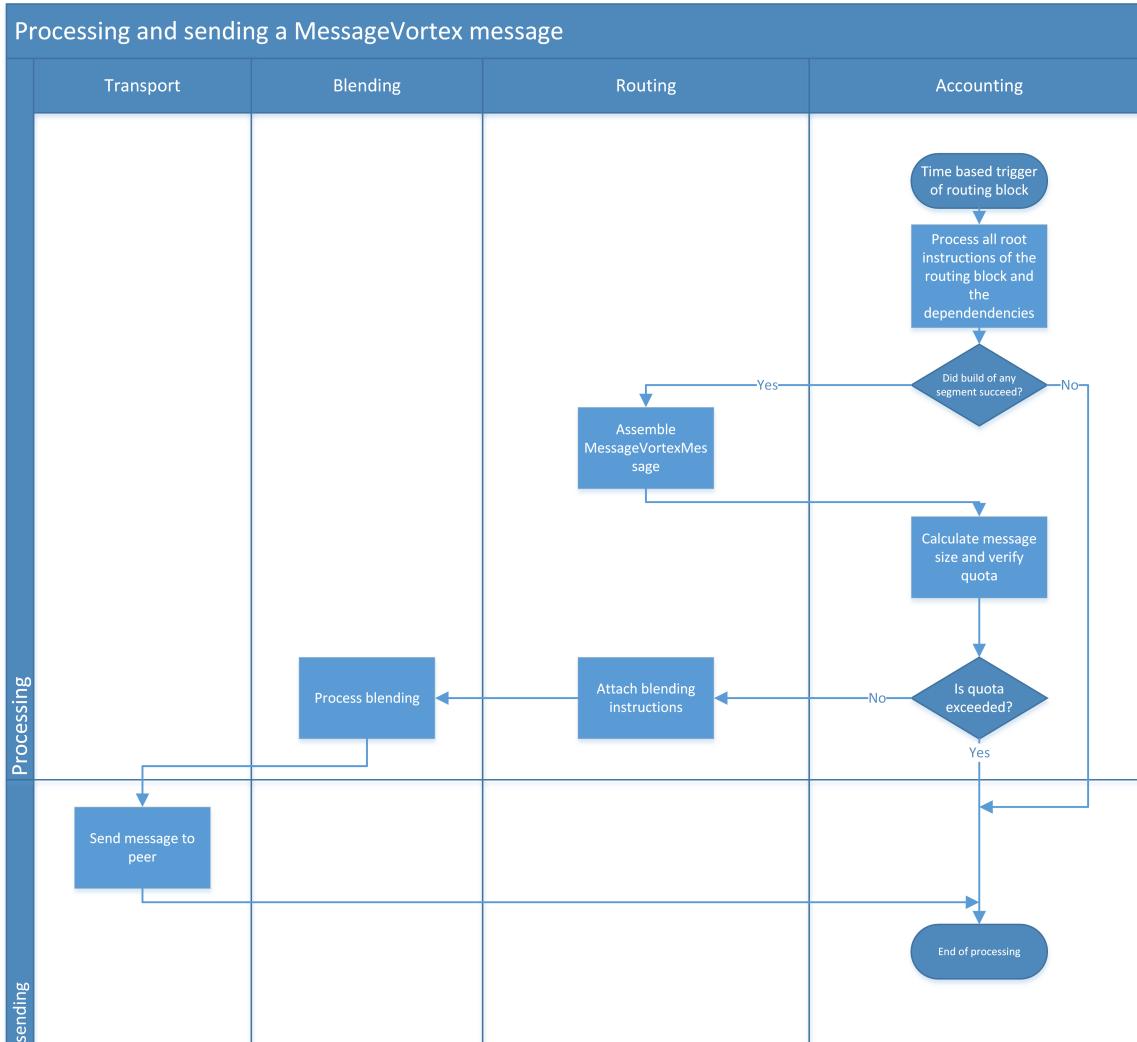


Figure 19.1: flow diagram showing processing of outgoing messages

19.3.2 Replacing an Existing Node Identity



19.3.3 Replacing an Existing Reply Block



20 Accounting Layer Implementation



21 Usability Related Implementation Details



21.1 Adressing and address representations



21.2 Linking to Common User Agents



22 Efficiency Related Implementation Details



22.1 Node Storage Management



22.1.1 Life-cycle of Ephemeral Identities and Workspaces



22.1.2 Life-cycle of Requests



22.1.3 Life-cycle of Operations



22.2 Overhead Minimization in Message



Operational concerns

Occurrences in this domain are beyond the reach of exact prediction because of the variety of factors in operation, not because of any lack of order in nature.

Albert Einstein



23 General Concerns Regarding Operation

23.1 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.

23.2 Addressing of Vortex Nodes

from the start we were looking for an addressing scheme suitable for transparent addressing

A Vortex address is built as follows:

```
1 localPart      = <local part of address>
2 domain        = <domain part of address>
3 email          = localPart "@" domain
4 keySpec        = <BASE64 encoded AsymmetricKey [DER encoded]>
5 smtpAlternateSpec = localPart "..." keySpec "..." domain "@localhost"
6 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 representation 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 clear-text 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

23.3 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.

23.3.1 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.

23.3.2 Vortex Node Types

23.3.2.1 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 ???. 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.

23.3.2.2 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.

23.3.2.3 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.

23.4 Strategies for Composing Routing Blocks

23.4.1 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.

23.5 Algorithms Suitable for Achieving Anonymity



23.6 Possibilities of Routing Diagnosis and Reputation Building



23.7 Possibilities of Redundancies



24 Protocol Bootstrapping



24.1 Key Distribution for Endpoints



24.2 Key Aquisition for Routing Nodes



25 Operational Concerns Regarding Environments with an Observing Adversary

26 Operational Concerns Regarding Environments with an Censoring Adversary



27 Real World Problems when using *MessageVortex*

27.1 Size Restrictions of the Transport Layer

A transport layer may limit the size of messages transferred. We managed to create *VortexMessages* as small as 2KB in size. Considering the blending overhead of F5 our message is sized at least 24KB which is not a problem for any of the selected transport protocol. So, while a *VortexMessage* may be small an upper size limit is

Analysis of MessageVortex

Atoms are very special: they like certain particular partners, certain particular directions, and so on. It is the job of physics to analyze why each one wants what it wants.

Richard P. Feynman

In section 13.1, we described two different kinds of adversaries. Which do require different properties for our system to be fulfilled.

An observing adversary is the less dangerous one. While observing all traffic he may not disrupt communication. Instead he is using all available information to collect data about all items of interest (IOI). He may do this for example by collecting inside or outside information about all message flows he may encounter. He may use this information and assign it to certain individuals or groups of individuals.

A censoring attacker is far more dangerous as he does not only observe the system, but he may systematically suppress freedom of speech and all technology related to it. As he has the means and the technical know-how, he may try, apart from observing, to discover systems communicating illegally either by observation or by infiltration of systems. He may furthermore track down individuals within its reach and prosecute them. All other participants of an illegal system may be either identified and blacklisted or even attacked either by infiltrating their systems or by effectively launching DoS attacks against those systems.

In the following sections, we will analyze aspects of security and highlight differences in terms of the different adversaries.

28 Analysis of the Blending Layer

29 Analysis of the Core Operations

29.1 Splitting and Merging

29.2 Encryption and Decryption Operations

29.3 Add and Remove Redundancy Operations



30 Well 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.

30.1 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 threats 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.

We have introduced a way to support a set of independent cryptographic algorithms, paddings, modes, and prngs. The support of these algorithms does not have to be uniform throughout the

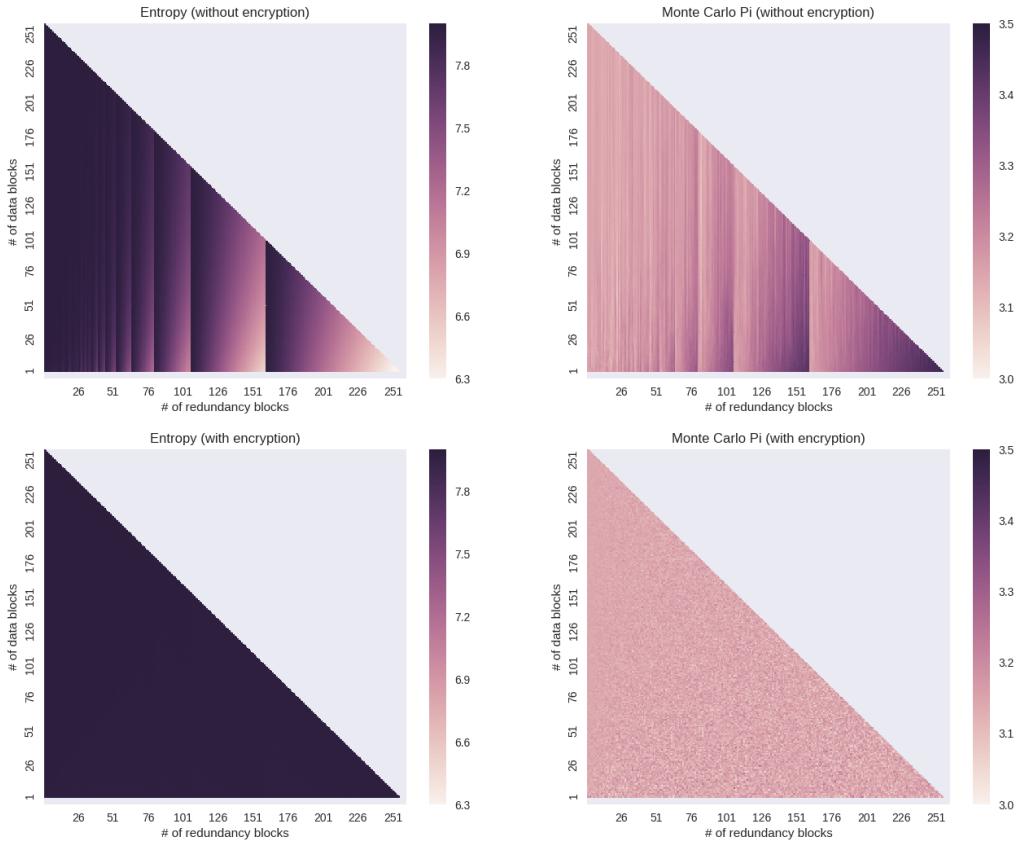


Figure 29.1: Resulting entropy of addRedundancy with and without encryption step

system, instead it is sufficient for two neighbouring nodes support the same algorithms in order to be used.

30.2 Attacks Targeting Anonymity

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

- Prevent an attack

Attack prevention can only be done for attacks that are already known and may not be realistic in all cases. In our protocol, we have strict boundaries defined. A node under attack should at any time of protocol usage (this excepts bandwidth depletion attacks) be able to block malicious identities. Since establishing new identities is costly for an attacker, he should always require far more resources than the defender.

- Minimize attack surface

This part of the attack prevention is included by design in the protocol.

- Redirect an attack

Although the implementation does not do this, it is possible to handle suspected malicious nodes differently.

- Control damage

For us, this means leaving as little information about identities or meta information as possible on untrusted infrastructures. If we leave traces (i.e., message flows, or accounting information) they should have the least possible information content and should expire within a reasonable amount of time.

- Discover an attack

The protocol is designed in such a way that attack discovery (such as a query attack) is possible. However, we consider active attacks just as part of the regular message flow. The protocol must mitigate such attacks by design.

- Recover from an attack

An attack does always impose a load onto a system's resources regardless of its success. It is vital that a system recovers almost immediately from an attack and is not covered in a non-functional or only partial-functional state either temporarily or permanently.

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

30.2.0.1 Probing Attacks

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

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

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

30.2.0.2 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.

30.2.0.3 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.

30.2.0.4 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.

30.2.0.5 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.

30.2.0.6 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.

30.2.1 Denial of Service Attacks

30.2.1.1 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.

30.2.1.2 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.

30.2.1.3 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.

31 Identification of Possible Attack Schemes and Mitigation



31.1 Static Attacks



31.1.1 Bugging and Tagging Attacks



31.1.2 Information Leaking related to Information Available to Routing Nodes



31.1.3 Identification of involved Nodes



31.1.4 Identification of MessageVortex Traffic



31.2 Dynamic Attacks



31.2.1 Attacks against the vortex system itself

31.2.1.1 DoS Attacks against the Transport System

31.2.1.2 DoS by Traffic Replay

31.2.1.3 DoS by Traffic generation

31.2.2 Attacking a single ephemeral Identity of a MessageVortex Node



31.2.2.1 Denial of Service by Exhausting Quotas or Limits

31.2.3 Attacking Sending and Receiving Identities of the MessageVortex System



31.2.3.1 Traffic Highlighting or Traffic Analysis

31.2.4 Recovery of Previously Carried Out Operations



32 Analysis of the effectiveness of Attack Schemes



33 Analysis of the Degree of Anonymization in Comparison to other Systems



34 To be placed (TBP)

34.1 Side Channel Leaking

34.1.1 Software Updates and Related Data Streams

34.1.2 Bugging in transported messages

Discussion on Results

Limit your inputs to only those that support a certain kind of self-destructive behavior, and you can be cheered with enthusiasm as you drive yourself off a cliff.

Adam-Troy Castro



35 Measuring up to the Requirements

In this section we analyze the level of achievement in respect to the requirements defined in section 13.2.1. We will go through each requirement and discuss the level of achievement. In the case of a failure, we highlight the reason for the failure and elaborate on the consequences of the current flaws.



36 Achieved level of anonymity



37 Weaknesses of the protocol



38 Further and Missing Research



Appendix

Limit your inputs to only those that support a certain kind of self-destructive behavior, and you can be cheered with enthusiasm as you drive yourself off a cliff.

Adam-Troy Castro

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

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

- [1. Introduction](#)
 - [1.1. Requirements Language](#)
 - [1.2. Protocol Specification](#)
 - [1.3. Number Specification](#)
- [2. Entities Overview](#)
 - [2.1. Node](#)
 - [2.1.1. Blocks](#)
 - [2.1.2. NodeSpec](#)
 - [2.2. Peer Partners](#)
 - [2.3. Encryption keys](#)
 - [2.3.1. Identity Keys](#)
 - [2.3.2. Peer Key](#)
 - [2.3.3. Sender Key](#)
 - [2.4. Vortex Message](#)
 - [2.5. Message](#)
 - [2.6. Key and MAC specifications and usage](#)
 - [2.6.1. Asymmetric Keys](#)
 - [2.6.2. Symmetric Keys](#)
 - [2.7. Transport Address](#)
 - [2.8. Identity](#)
 - [2.8.1. Peer Identity](#)
 - [2.8.2. Ephemeral Identity](#)
 - [2.8.3. Official Identity](#)
 - [2.9. Workspace](#)

2.10. Multi-use Reply Blocks

3. Layer Overview

3.1. Transport Layer

3.2. Blending Layer

3.3. Routing Layer

3.4. Accounting Layer

4. Vortex Message

4.1. Overview

4.2. Message Prefix Block (MPREFIX)

4.3. Inner Message Block

4.3.1. Control Prefix Block

4.3.2. Control Blocks

4.3.3. Payload Block

5. General notes

5.1. Supported Symmetric Ciphers

5.2. Supported Asymmetric Ciphers

5.3. Supported MACs

5.4. Supported Paddings

5.5. Supported Modes

6. Blending

6.1. Blending in Attachments

6.1.1. PLAIN embedding into attachments

6.1.2. F5 embedding into attachments

6.2. Blending into an SMTP layer

6.3. Blending into an XMPP layer

7. Routing

7.1. Vortex Message Processing

7.1.1. Processing of incoming Vortex Messages

7.1.2. Processing of Routing Blocks in the Workspace

7.1.3. Processing of Outgoing Vortex Messages

7.2. Header Requests

7.2.1. Request New Ephemeral Identity

7.2.2. Request Message Quota

7.2.3. Request Increase of Message Quota

7.2.4. Request Transfer Quota

7.2.5. Query Quota

7.2.6. Request Capabilities

7.2.7. Request Nodes

7.2.8. Request Identity Replace

7.3. Special Blocks

7.3.1. Error Block

7.3.2. Requirement Block

7.4. Routing Operations

7.4.1. Mapping Operation

7.4.2. Split and Merge Operations

7.4.3. Encrypt and Decrypt Operations

7.4.4. Add and Remove Redundancy Operations

7.5. Processing of Vortex Messages

[8. Accounting](#)

[8.1. Accounting Operations](#)

[8.1.1. Time-Based Garbage Collection](#)

[8.1.2. Time-Based Routing Initiation](#)

[8.1.3. Routing Based Quota Updates](#)

[8.1.4. Routing Based Authorization](#)

[8.1.5. Ephemeral Identity Creation](#)

[9. Acknowledgments](#)

[10. IANA Considerations](#)

[11. Security Considerations](#)

[12. References](#)

[12.1. Normative References](#)

[12.2. Informative References](#)

[Appendix A. The ASN.1 schema for Vortex messages](#)

[A.1. The main VortexMessageBlocks](#)

[A.2. The VortexMessage Ciphers Structures](#)

[A.3. The VortexMessage Request Structures](#)

[A.4. The VortexMessage Replies Structures](#)

[A.5. The VortexMessage Requirements Structures](#)

[A.6. The VortexMessage Helpers Structures](#)

[A.7. The VortexMessage Additional Structures](#)

[Author's Address](#)

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 [[CCITTX208.1988](#)]). 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](#).

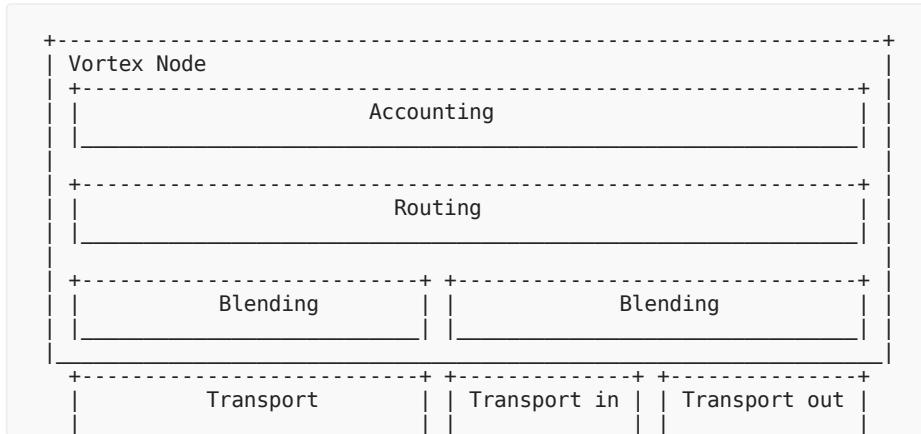


Figure 1: Layer overview

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.

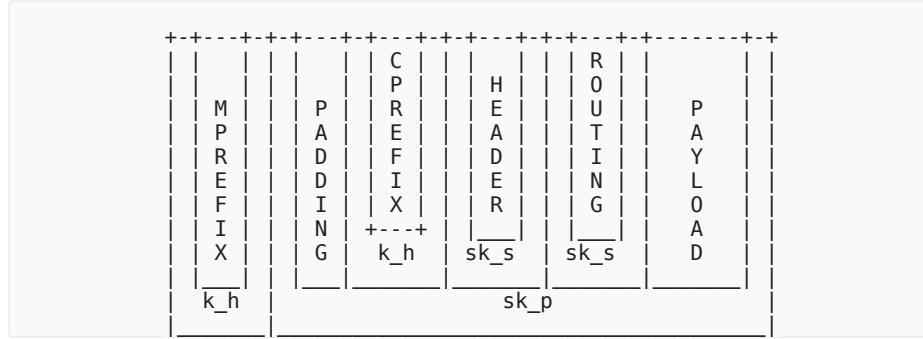


Figure 2: Vortex message overview

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:" <numberOfBytesOffset>  
[ "," <numberOfBytesOffset> ]* ")"
```

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 incoming_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 $L=\text{roof}((\text{input block size in bytes})+4)/(\text{encryption block size in bytes})^*\text{encryption block size in bytes}$. The block is prepended with a 32-bit unit length indicator in bytes (little-endian). This length indicator, i , is calculated by $i=\text{input block size in bytes}*\text{randominteger}\cdot L$. 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 $\text{output block size in bytes}=i \bmod \text{input block size in bytes}$

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 $v(i,j)=\text{pow}(i,j)$, 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 Encryption 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 identifier 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
    encrypted [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
    encrypted [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),  
  
        -- Major 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,  
        amount         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),  
        xmpp (110)  
    }  
  
END
```

A.7. The VortexMessage Additional Structures

```

-- States: Tuple()=Value() [validity; 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 omitted own identity key
nodeAddress        [1002] NodeSpec      OPTIONAL,
-- Contains the identity of the owning node;
-- May be omitted 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](tel:+41562027681)
Email: rfc@messagevortex.net

B Glossary

adversary In this work, we are referring to an adversary to any entity opposing to the privacy of a message. For a more throughout definition refer to section 13.1

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.

Decoy traffic Any data transported between routers that have no relevance to the message at the final destination and are not needed for the flow of the message.

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.

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.

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.

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. An Identity is always assigned to a node, but one node may have multiple identities.

Jurisdiction 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. Most of these legal rules are based on their specific physical location (e.g., German law is limited to the jurisdiction of Germany). Some jurisdictions may over-arch multiple separated geographical locations (e.g., laws of the European Union) or specific to some handpicked countries (e.g., International Covenant on Civil and Political Rights). Due to the overlapping nature multiple jurisdictions may have contradictory rules applying for the same event.

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 Interest (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.

¹footnotes omitted in quote

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.

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.

Message The “real content” to be transferred from the sender to the recipient. Please note the difference compared to a vortex message. We refer to the encoded form of a VortexMessage, which may or may not contain parts of the original message as VortexMessage.

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 possiblitz 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.

Payload Any data transported between routers regardless of the meaningfulness or relevance to the VortexMessage.

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 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

Recipient The user or process destined to receive the message in the end.

Router Any VortexNode which is processing messages. Please note that all VortexNodes are routers.

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.

Sender The user or process originally composing the message. we refer as the sender both the human creator or initiator of a message, as well as the process assembling and preparing the message.

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.

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 [\[anonTerminology\]](#).

²footnotes omitted in quote

Undetectability of an item of interest (IOI) from an attacker's perspective means that the attacker cannot sufficiently distinguish whether it exists or not.³

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”):

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

User Any entity operating a *VortexNode*.5

VortexMessage The encoded message passed from one VortexNode to another one. The VortexMessage is typically considered before any embedding takes place.

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.

³footnotes omitted in quote

B Bibliography

B 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.

