



MessageVortex

Transport Independent Messaging anonymous to 3rd Parties

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

October 28, 2018

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



Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
Auf Antrag von

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

Basel, der 17.10.2017 durch die Fakultätsversammlung

Prof. Dr. Jörg Schibler

Abstract

In this paper we introduce an unobservable message anonymisation protocol, named MessageVortex. It is based on the zero trust principle and 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. It furthermore requires no protocol specific infrastructure in public networks and allows a sender to control all aspects of a message such as degree of anonymity, timing, and redundancy of the message transport without disclosing any of these details to the routing or transporting nodes.

Contents

| | | |
|--------|--|----|
| 1. | Introduction | 1 |
| 1.1. | Contributions | 2 |
| 1.2. | Notation | 2 |
| 1.2.1. | Cryptography | 2 |
| 1.2.2. | Code and commands | 3 |
| 1.2.3. | Hyperlinking | 3 |
| 2. | Main Research Question | 5 |
| 2.1. | SQ1: Sending messages maintaining unlinkability against an adversary | 5 |
| 2.2. | SQ2: Attacking unlinkability and circumvention | 5 |
| 2.3. | SQ3: Attack Mitigation by design | 5 |
| I. | Methodes | 7 |
| 3. | Requirements for an anonymising Protocol | 11 |
| 3.1. | Anonymizing and Unlinking | 11 |
| 3.2. | Censorship Resistant | 12 |
| 3.3. | Controllable trust | 12 |
| 3.4. | Reliable | 12 |
| 3.5. | Diagnosable | 12 |
| 3.6. | Available | 12 |
| 3.7. | Rough draft of Protocol Layers | 13 |
| 3.8. | Rough Draft of Messages | 14 |
| 4. | Existing Transport Layer Protocols | 15 |
| 4.1. | HTTP | 15 |
| 4.2. | FTP | 15 |
| 4.3. | TFTP | 16 |
| 4.4. | MQTT | 16 |
| 4.5. | Advanced Message Queuing Protocol | 16 |
| 4.6. | Constrained Application Protocol (CoAP) | 17 |
| 4.7. | Web Application Messaging Protocol | 17 |
| 4.8. | XMPP (jabber) | 17 |
| 4.9. | SMTP | 17 |
| 4.10. | SMS | 18 |
| 4.11. | MMS | 18 |
| 4.12. | Roundup for Transport Protocols | 18 |
| 5. | Existing Research and Implementations on the Topic | 19 |
| 5.1. | Anonymity | 19 |
| 5.1.1. | k -Anonymity | 19 |
| 5.1.2. | ℓ -Diversity | 19 |
| 5.1.3. | t -Closeness | 19 |
| 5.2. | Zero Trust | 19 |
| 5.3. | Pseudonymity | 20 |
| 5.4. | Undetectability | 20 |
| 5.5. | Unobservability | 20 |
| 5.5.1. | Ephemeral Identity | 20 |
| 5.6. | Single Use Reply Blocks and Multi Use Reply Blocks | 20 |
| 5.7. | Censorship | 21 |
| 5.7.1. | Censorship Resistant | 21 |

| | | |
|----------------|--|----|
| 5.7.2. | Parrot Circumvention | 21 |
| 5.7.3. | Censorship Circumvention | 21 |
| 5.7.3.1. | Covert Channel and Channel Exploitations | 21 |
| 5.7.3.2. | Spread Spectrum | 22 |
| 5.7.3.3. | Steganography | 22 |
| f5 | | 22 |
| YASS | | 22 |
| 5.7.3.4. | Timing Channels | 22 |
| 5.8. | Cryptography | 22 |
| 5.8.1. | Symmetric Encryption | 22 |
| 5.8.1.1. | AES | 22 |
| 5.8.1.2. | Camellia | 22 |
| 5.8.2. | Asymmetric Encryption | 22 |
| 5.8.2.1. | RSA | 23 |
| 5.8.2.2. | Elliptic Curve Cryptography | 23 |
| 5.8.3. | Homomorphic encryption | 23 |
| 5.8.4. | Deniable Encryption and Deniable Steganography | 23 |
| 5.8.5. | Key Sizes | 23 |
| 5.8.6. | Cypher Mode | 23 |
| 5.8.6.1. | ECB (Electronic Code Book): | 24 |
| 5.8.6.2. | CBC (Cypher Block Chaining): | 24 |
| 5.8.6.3. | PCBC (Propagation Cypher Block Chaining): | 24 |
| 5.8.6.4. | EAX: | 24 |
| 5.8.6.5. | CFB (Cypher Feedback): | 24 |
| 5.8.6.6. | OFB: | 24 |
| 5.8.6.7. | OCB (Offset Codebook Mode): | 24 |
| 5.8.6.8. | CTR: | 24 |
| 5.8.6.9. | CCM: | 25 |
| 5.8.6.10. | GCM (Galois Counter Mode): | 25 |
| 5.8.7. | Padding | 25 |
| 5.8.7.1. | PKCS1: | 25 |
| 5.8.7.2. | PKCS7: | 25 |
| 5.8.7.3. | OAEP with SHA and MGF1 padding: | 25 |
| 5.9. | Routing | 26 |
| 5.9.1. | Mixing | 26 |
| 5.9.2. | Onion Routing | 26 |
| 5.9.3. | Crowds | 26 |
| 5.9.4. | Mimic routes | 27 |
| 5.9.4.1. | DC Networks | 27 |
| 5.9.4.2. | Anonymous Remailer | 27 |
| 5.10. | System Implementations | 28 |
| 5.10.1. | Pseudonymous Remailer | 28 |
| 5.10.2. | Babel | 28 |
| 5.10.3. | Cypherpunk-Remailer | 28 |
| 5.10.4. | Mixmaster-Remailer | 28 |
| 5.10.5. | Mixminion-Remailer | 28 |
| 5.10.6. | Crowds | 28 |
| 5.10.7. | Tarzan | 28 |
| 5.10.8. | Tor | 28 |
| 5.10.9. | <i>I²P</i> | 29 |
| 5.10.10. | Freenet | 29 |
| 5.10.11. | Herbivore | 30 |
| 5.10.12. | Dissent | 30 |

| | |
|--|----|
| 5.10.13. \mathcal{P}^5 | 30 |
| 5.10.14. Gnutella | 30 |
| 5.10.15. Gnutella2 | 30 |
| 5.10.16. Hordes | 30 |
| 5.10.17. Salsa | 30 |
| 5.10.18. AP3 | 30 |
| 5.10.19. Hydra-Onion | 30 |
| 5.10.20. DUO-Onion | 30 |
| 5.10.21. Cashmere | 31 |
| 5.10.22. SMTP | 31 |
| 5.10.23. Mail Endpoints | 32 |
| 5.10.23.1. Fat clients | 32 |
| 5.10.23.2. Server located clients | 33 |
| 5.10.23.3. Web clients | 33 |
| 5.10.24. Interfaces of Mail Endpoints | 33 |
| 5.10.25. S/MIME | 33 |
| 5.10.26. PGP/MIME | 33 |
| 5.10.27. XMMP | 33 |
| 5.11. Proof of work | 33 |
| 5.11.1. CPU-Bound Proof of work | 34 |
| 5.11.1.1. Hash based puzzle by Juels and Brainard | 34 |
| 5.11.1.2. Hash based puzzle by Aura, Nikander and Leiwo | 34 |
| 5.11.1.3. Modified Repeated Squaring Puzzle | 34 |
| 5.11.1.4. Hinted Hash Puzzle | 35 |
| 5.11.1.5. Chained Hash Puzzle | 35 |
| 5.11.1.6. Targeted Hash Puzzle | 35 |
| 5.11.1.7. Repeated Squaring Puzzle | 35 |
| 5.11.1.8. Discrete Logarithm Puzzle | 35 |
| 5.11.1.9. Subset sum puzzle | 36 |
| 5.11.2. Memory-Bound Proof of work | 36 |
| 5.11.2.1. Moderately hard, memory-bound functions | 36 |
| 5.11.2.2. Memory bound puzzle based on pattern databases | 36 |
| 5.11.2.3. Argon2 | 36 |
| 5.11.2.4. Scrypt | 36 |
| 5.11.2.5. HashCash | 36 |
| 5.11.3. Summary for Proof of Work Algorithms | 36 |
| 5.12. Pseudo Random Number Generators | 36 |
| 5.13. Known Attacks | 36 |
| 5.13.1. Broken Encryption Algorithms | 37 |
| 5.13.2. Attacks Targeting Anonymity | 37 |
| 5.13.2.1. Probing Attacks | 37 |
| 5.13.2.2. Hotspot Attacks | 37 |
| 5.13.2.3. Message Tagging and Tracing | 38 |
| 5.13.2.4. Side Channel Attacks | 38 |
| 5.13.2.5. Timing Attacks | 38 |
| 5.13.2.6. Sizing Attacks | 38 |
| 5.13.2.7. Bugging Attacks | 38 |
| Bugging through certificate or identity lookup: | 38 |
| Bugging through DNS traffic: | 38 |
| Bugging through external resources: | 38 |
| 5.13.3. Denial of Service Attacks | 39 |
| 5.13.3.1. Censorship | 39 |
| 5.13.3.2. Credibility Attack | 39 |

| | | |
|----------|--|----|
| 6. | Applied Methodes | 41 |
| 6.1. | Problem Hotspots | 41 |
| 6.1.1. | Zero Trust Philosophy | 41 |
| 6.1.2. | Information leakage | 41 |
| 6.1.2.1. | P2P design | 41 |
| 6.1.2.2. | Decoy traffic generation | 41 |
| 6.1.2.3. | Message tagging or bugging protection | 42 |
| 6.1.2.4. | Message replay protection | 42 |
| 6.1.2.5. | No Dedicated Infrastructure Philosophy | 42 |
| 6.1.3. | Accounting | 42 |
| 6.1.4. | Anonymisation | 43 |
| 6.1.5. | Initial Bootstrapping | 43 |
| 6.1.6. | Cypher selection | 43 |
| 6.1.7. | Reed-Solomon function | 44 |
| 6.1.8. | Usability | 44 |
| 6.2. | Protocol outline | 44 |
| 6.2.1. | Protocol Terminology | 44 |
| 6.2.2. | Vortex Communication model | 45 |
| 6.2.3. | Transport Layer | 45 |
| 6.2.4. | Blending Layer | 45 |
| 6.2.5. | Routing Layer | 45 |
| 6.2.5.1. | Block Structure | 46 |
| 6.2.5.2. | MURBs | 46 |
| 6.3. | Protocol design | 46 |
| 6.3.1. | Header block | 46 |
| 6.3.1.1. | Ephemeral Identity | 47 |
| 6.3.1.2. | Requests | 47 |
| | newIdentity request: | 47 |
| | queryPeer request: | 47 |
| | queryCapability request: | 48 |
| | messageQuota request: | 48 |
| | transferQuota request: | 48 |
| | queryQuota request: | 48 |
| 6.3.1.3. | Replies to Clear Text Requests | 48 |
| 6.3.2. | Main block | 48 |
| 6.3.2.1. | Routing Blocks | 48 |
| 6.3.2.2. | Payload Building Instructions | 48 |
| | splitPayload: | 49 |
| | mergePayload: | 49 |
| | xorMergePayload: | 49 |
| | xorSplitPayload: | 49 |
| | encryptPayload: | 49 |
| | decryptPayload: | 49 |
| 6.3.2.3. | Reply Block | 49 |
| | replyCapability block: | 49 |
| | replyPuzzleRequired block: | 49 |
| | replyStatus block: | 49 |
| | ctxlessNewidentity block: | 50 |
| 6.3.2.4. | payload Block | 50 |
| 6.3.3. | Accounting | 50 |
| 6.3.3.1. | Accounting Data of an Identity | 50 |
| 6.3.3.2. | Accounting Data of a “header” Block | 51 |
| 6.3.3.3. | Accounting Data of a “payload” Block | 51 |

| | | |
|----------|--|----|
| 6.4. | Protocol handling | 51 |
| 6.4.1. | Received Block Processing | 51 |
| 6.4.2. | Transmission Block Processing | 51 |
| 6.5. | Sub Research Questions Roundup | 51 |
| 6.5.1. | SQ1: Sending messages maintaining unlinkability against an adversary | 51 |
| 6.5.2. | SQ2: Attacking unlinkability and circumvention | 51 |
| 6.5.3. | SQ3: Attack Mitigation by design | 51 |
| II. | Results | 53 |
| 7. | Protocol Overview | 57 |
| 7.1. | Vortex Message | 57 |
| 7.1.1. | Key Usage | 58 |
| 7.1.1.1. | Peer key | 58 |
| 7.1.1.2. | Header Key | 58 |
| 7.1.1.3. | Identity Key | 58 |
| 7.1.1.4. | Ephemeral Identity Key | 59 |
| 7.1.2. | VortexMessage Processing | 59 |
| 7.1.2.1. | Receiving Messages | 59 |
| 7.1.2.2. | Building and Sending Messages | 59 |
| 7.1.3. | VortexMessage Operations | 59 |
| 7.1.3.1. | VortexMessage SplitPayload Operation | 59 |
| 7.1.3.2. | VortexMessage MergePayload Operation | 60 |
| 7.1.3.3. | VortexMessage XorSplitPayload Operation | 60 |
| 7.1.3.4. | VortexMessage XorMergePayload Operation | 60 |
| 7.1.3.5. | VortexMessage EncryptPayload Operation | 60 |
| 7.1.3.6. | VortexMessage DecryptPayload Operation | 61 |
| 7.1.3.7. | VortexMessage addRedundancy Operation | 61 |
| 7.1.3.8. | VortexMessage removeRedundancy Operation | 61 |
| 7.2. | VortexMessage Request Processing | 61 |
| 7.2.1. | VortexMessage Requests | 61 |
| 7.2.1.1. | QueryCapability Request | 61 |
| 7.2.1.2. | NewIdentity Request | 61 |
| 7.2.1.3. | QueryPeer Request | 62 |
| 7.2.1.4. | TransferQuota, MessageQuota, and QueryQuota Request | 62 |
| 7.2.2. | VortexMessage Reply Blocks | 62 |
| 7.2.2.1. | ReplyCapability block | 62 |
| 7.2.2.2. | replyPuzzleRequired block | 62 |
| 7.2.2.3. | replyStatus block | 62 |
| 7.2.2.4. | CtxlessNewidentity block | 63 |
| 8. | Vortex Prerequisites | 65 |
| 8.1. | Hardware | 65 |
| 8.1.1. | Considerations Regarding Weak or Unsuitable Hardware | 65 |
| 8.2. | Addresses | 65 |
| 8.2.1. | Public Key Encoding in Address Representation | 65 |
| 8.3. | Transport Layers | 65 |
| 8.3.1. | SMTP | 65 |
| 8.3.2. | XMMP | 65 |
| 8.4. | Client | 66 |
| 8.4.1. | Vortex Accounts | 66 |
| 8.4.2. | Vortex Node Types | 66 |
| 8.4.2.1. | Public Vortex Node | 66 |
| 8.4.2.2. | Stealth Vortex Node | 66 |
| 8.4.2.3. | Hidden Vortex Node | 66 |

| | | |
|----------|--|----|
| 9. | MessageVortex - Transport Independent Messaging anonymous to 3 rd Parties | 67 |
| 9.1. | Protocol Implementation | 67 |
| 9.2. | Block Structure | 67 |
| 9.2.1. | Header Block | 67 |
| 9.2.2. | Routing Blocks | 67 |
| 9.2.3. | Payload Blocks | 67 |
| 9.3. | Message Building | 67 |
| 9.4. | Accounting | 67 |
| 9.5. | Blending layer | 67 |
| 9.5.1. | Plain Inclusion | 68 |
| 9.5.1.1. | File Type Candidates | 68 |
| 9.5.2. | F5 Steganography | 68 |
| 9.6. | Message Flows | 68 |
| 9.7. | Considerations for Building Messages | 68 |
| 9.7.1. | Ephemeral identities | 69 |
| 9.7.2. | Timing of messages | 69 |
| 9.7.3. | Diagnostics | 69 |
| 9.7.3.1. | Implicit Diagnostic | 69 |
| 9.7.3.2. | Automatic Explicit Diagnostic | 69 |
| 9.7.3.3. | On-Demand Explicit Diagnostic | 69 |
| 9.8. | Verification of requirements | 69 |
| 9.9. | Considerations for Routing Messages | 70 |
| 9.9.1. | Time of sending | 70 |
| 9.10. | Real World Considerations | 70 |
| 9.10.1. | No Routing Log | 70 |
| 9.10.2. | No Verification Possibilities | 71 |
| 9.10.3. | Very Limited Control Blending Layer for the sending node | 71 |
| 9.10.4. | Message Content | 71 |
| 10. | Security Analysis | 73 |
| 10.1. | Attack on Users Identity | 73 |
| 10.1.1. | Frequency and Bandwidth Analysis | 73 |
| 10.2. | Attack on Message content | 73 |
| 10.2.1. | Attacking Routing Blocks | 73 |
| 10.3. | Attack on Message size | 73 |
| 11. | Additional Considerations | 75 |
| 11.1. | Man in the Middle Attacks to Conversations | 75 |
| 11.2. | Identification of Participating Accounts | 75 |
| 11.2.1. | Identification by Content | 75 |
| 11.2.2. | Identification by Query | 75 |
| 11.2.3. | Identification by Traffic Type | 75 |
| 11.2.4. | Identification by Bandwidth | 75 |
| 11.2.5. | Identification by Behavioural Analysis | 75 |
| 11.3. | Storage of Messages and queues | 75 |
| 11.4. | Economy of transfer | 75 |
| III. | Discussion | 77 |
| 12. | Static analysis | 81 |
| 12.1. | Transport and Blending Layer | 81 |
| 12.1.1. | Identifying a Vortex Message Endpoint | 81 |
| 12.2. | Senders routing layer | 81 |
| 12.3. | Intermediate node routing layer | 81 |
| 12.4. | Receivers routing layer | 81 |

| | |
|--|------|
| 13. Dynamic analysis | 83 |
| 13.1. Attacks against the vortex system itself | 83 |
| 13.1.1. DoS Attacks against the System | 83 |
| 13.1.1.1. Traffic Replay | 83 |
| 13.1.2. Diagnosability of traffic | 83 |
| 13.1.2.1. Hijacking of Header and Routing Blocks | 83 |
| 13.1.2.2. Partial Implicit Routing Diagnosis | 83 |
| 13.1.2.3. Partial Explicit Routing Diagnosis | 83 |
| 13.1.2.4. End-to-End Routing Diagnosis | 83 |
| 13.1.2.5. Denial of Service by Exhausting Quotas or Limits | 83 |
| 13.2. Achieved Anonymity and Flaws | 84 |
| 13.2.1. Measuring Anonymity | 84 |
| 13.2.2. Attacking Anonymity through Traffic Analysis | 84 |
| 13.2.2.1. Attacking Anonymity through Timing Analysis | 84 |
| 13.2.2.2. Attacking Anonymity through Throughput Analysis | 84 |
| 13.2.3. Attacking Anonymity through Routing Block Analysis | 84 |
| 13.2.4. Attacking Anonymity through Header Analysis | 84 |
| 13.2.5. Attacking Anonymity through Payload Analysis | 84 |
| 13.2.6. Attacking Anonymity through Bugging | 84 |
| 13.2.7. Attacking Anonymity through Replay Analysis | 84 |
| 13.2.8. Attacking Anonymity through Tamper Replay Analysis | 84 |
| 14. Recommendations on using the Vortex Protocol | 85 |
| 15. Achieved Goals and Weaknesses | 87 |
| 15.1. Message content | 87 |
| 15.1.1. Splitting of message content | 87 |
| 15.1.2. Redundancy | 87 |
| 15.1.3. Redundancy Detection as Attack Pattern | 87 |
| 15.1.4. routing Considerations | 87 |
| 15.1.5. Hotspot Avoidance | 87 |
| 16. Anonymity | 89 |
| 16.0.1. Size of the Anonymity Set | 89 |
| 16.0.2. Jurisdictional implications onto the Anonymity Set | 89 |
| 16.1. Effects of anonymous communication on behaviour | 89 |
| IV. Appendix | A1 |
| The RFC draft document | A3 |
| Glossary | A51■ |
| Bibliography | A53■ |
| Short Biography | A61■ |

List of Tables

| | |
|--|----|
| 4.1. comparison of protocols in terms of the suitability criteria as transport layer | 18 |
| 5.1. comparison of encryption modes in terms of the suitability criteria as transport layer . | 25 |
| 5.2. comparison of Proof of Work algorithms | 35 |
| 9.1. Requests and the applicable criteria for replies | 68 |

List of Figures

| | | |
|-------|---|----|
| 3.1. | A traceroute to the host www.ietf.org | 11 |
| 3.2. | A first rough overview for the protocol | 13 |
| 5.1. | Mail Agents | 31 |
| 6.1. | flow diagram showing processing of incomming messages | 52 |
| 7.1. | Simplified protocol outline | 58 |
| 9.1. | Overview of the Vortex modules | 68 |
| 12.1. | Detailed representation of a VortexMessage | 82 |

List of Requirements

| | |
|--|----|
| RQ1 No infrastructure should be trusted unless it is the senders or the recipients infrastructure. | 41 |
| RQ2 Mixes and peers must be undistinguishable. | 41 |
| RQ3 The message should be untagable (neither by a sender, or by an intermediate party such as a mixer). | 42 |
| RQ4 The message should be unbugable (neither by a sender, or by an intermediate party such as a mixer). | 42 |
| RQ5 A message must not be replayable | 42 |
| RQ6 The system must be able to do accounting without being linked to a real identity. | 43 |
| RQ7 System must be able to anonymise sender and recipient at any point of the transport layer and at any point of mixing unless it is the sender or the recipient itself. | 43 |
| RQ8 The system must allow to boot strap from a zero knowledge or near zero knowledge point and extend the network on its own. | 43 |
| RQ9 The system must be able to use multiple symmetric, asymmetric, and hashing algorithms in order to immediately fall back to a secure algorithm for all new messages if required. | 43 |
| RQ10 The system must be usable without cryptographic know-how and with well known tools. | 44 |

1. Introduction

Almon Brown Strowger was owner of a funeral parlour in St. Petersburg. He filed a patent in March 10th 1891 for an “Automatic Telephone Exchange” [98]. 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 it was replaced by tone dialing.

This dialing technology enabled automatic messaging for voice and text messages (e.g. telex) up until today and built the base technology for today’s routed networks.

These networks are building the base for our communication-based society today. We are using these networks today as a common communication meaning for all purposes and do spend very little thoughts on the possible consequences arising if someone is able to put hands on this communication. This is not only a question if someone wants to hide something. The main problem is that the broad availability of data enabled our society to judge on peoples not only on what they are doing but as well, on what they did. While this is nothing bad people seem to be unable to differentiate that things that might have been common and normal in the past might be judged differently in present.

Numerous events in present and past show that multiple actors, some of which are state sponsored, collected data on a broad base in the internet. Whether this is a problem or not may be a disputable fact. Undisputed is however that if data is not handled with care peoples are accused with numerous “facts” that are more than questionable. To show that this may happen even under complete democratic control 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 (cover-

ing roughly 10% of the natural and juristic entities within Switzerland at that time). More about the Fichenskandal may be found in the Swiss Federal Archives (<https://www.bar.admin.ch>)

A series of similar attempts to attack privacy on a global scale have been discovered by whistleblower Edward Snowden. 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 to prove claims based on these documents. However – because the documents were screened by a significant number of journalists spanning multiple countries, 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 member of governments even in associated states (such as the german president mobile phone). 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.

This list of events shows that big players are collecting and storing vast amounts of data for analysis or possibly future use. The list of events shows also that the use of this data has in the past been at least partially questionable. As a part of possible counter measures 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 vast 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[43] or PGP[35]) 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 informations such as “message subject”, “frequency of exchanged messages”, “size

of messages”, or “client being used”. A good anonymity protocol has therefore far more attributes to cover than the message itself. It includes beside the message itself, all metadata, and all the traffic flows. Furthermore a protocol anonymising messages should not rely on the trust of infrastructure other than the infrastructure under control of the sending or receiving entity as a trust in any third party might be misleading.

Furthermore – Any central infrastructure is bound to be of special interest to anyone gathering data concerning the using entities of such a protocol. It furthermore may be manipulated in order to attack the messages or their flow. So central infrastructure have to be avoided.

In this work a new protocol is designed to allow 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 in short just “Vortex”. The protocol is designed in such a way so that it is capable to use a wide variety of transfer protocols. It is even possible to switch protocols while the messages are in transfer. This allows media breaches (at least on a protocol level) and makes analysis even harder.

The new protocol allows safe communication without the need of trusting the underlying transport media. Furthermore the usage of the protocol itself is possible without altering the immediate behaviour of the transport layer. That way it is possible to use the transport layers normal traffic to increase the noise in which information has to be searched.

This work splits into multiple parts. In the first part we collect available researches and technologies. We emphasize in all technologies on the strength and weaknesses relevant to this work. In the second part we reassemble the parts to a new protocol. In the third part we analyse the protocol for fitness of the purpose. We try to find weaknesses and work out recommendations for the protocol usage. In the last part we discuss the results and try to summarize the findings. We furthermore elaborate to what extend the protocol fulfils the requirements mentioned in the previous sections.

1.1. Contributions

This thesis contributes to the topic in the following senses:

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

1.2. Notation

1.2.1. Cryptography

The theory in this document is heavily based on symmetric encryption, asymmetric encryption and hashing. In order to use a uniformed notation I use $E^{K_a}(M)$ (where a is an index to distinguish multiple keys) resulting in M^{K_a} as the encrypted message. If we are reflecting a tuple of information we write it in boldface. to express the tuples content we use angular brackets $L\langle \text{normalAddress}, \text{vortexAddress} \rangle$. 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 M with a key K_a resulting in M^{K_a} where a is an index to distinguish different keys. Decryption uses therefore $D^{K_a}(M^{K_a}) = M$.

As notation for asymmetric encryption we use $E^{K_a^1}(M)$ 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 $D^{K_a^{-1}}(M)$.

For hashing we do use $H(M)$ 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 $M\langle t, \text{MURB}, \text{serial} \rangle$ respectively if encrypted $M^{K_a}\langle t, \text{MURB}, \text{serial} \rangle$.

| | | |
|---------------------|---------------------------------------|---------------------------|
| asymmetric: | $E^{K_a^{-1}}(\mathbf{M})$ | $= \mathbf{M}^{K_a^{-1}}$ |
| | $D^{K_a^1}(E^{K_a^{-1}}(\mathbf{M}))$ | $= \mathbf{M}$ |
| | $D^{K_a^{-1}}(E^{K_a^1}(\mathbf{M}))$ | $= \mathbf{M}$ |
| symetric: | $E^{K_a}(\mathbf{M})$ | $= \mathbf{M}^{K_a}$ |
| | $D^{K_a}(E^{K_a}(\mathbf{M}))$ | $= \mathbf{M}$ |
| 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 values of the same type and superscript denote relevant parameters to operations expressed. The subscripted and superscripted information may be omitted where not needed.

We refer to the components of a Vortex Message 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 | $= D^{K_a^1}(\mathbf{R}^{K_a^{-1}}) = D(\mathbf{R})$ |

```
# 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.
```

1.2.3. Hyperlinking

The electronic version of this document is hyper-linked. This means that references to the glossary or the literature may be clicked to find the respective entry. Chapter or table references are clickable too.

In general a decrypted Block is written as capitalized multi character boldface. An encrypted Block is written as capitalized single character boldface.

1.2.2. Code and commands

Code blocks are always displayed as light grey block with line numbers:

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

Commands entered at the command line are in a grey box with 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 have to be broken to fit into the paper a “ \leftarrow ” is inserted to indicate that the line break has been introduced for readability.

2. Main Research Question

The main topic of this thesis was defined as follows:

- Is it possible to have specialized messaging protocol used in 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 main question there are several sub questions grouped around various topics:

1. What technologies and methods may be used to provide sender and receiver anonymity and unlinkability when sending messages against a potential adversary? (SQ1)
2. How can entities utilizing MessageVortex be attacked and what measures are available to circumvent such attacks? (SQ2)
3. How can attacks targeting anonymity of a sending or receiving entity be mitigated by design within MessageVortex? (SQ3)

2.1. SQ1: Sending messages maintaining unlinkability against an adversary

This question covers the principal part of the work. We try to elaborate first rough criteria for the MessageVortex protocol. We then create a list of suitable technologies. Based on this list we define a protocol combining these technologies and researches to a solution. This solution will be implemented and analysed for suitability based on the criteria defined.

2.2. SQ2: Attacking unlinkability and circumvention

Within this questions we look at common 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 try to elaborate if these attacks might be successful (and if so under what circumstances).

2.3. SQ3: Attack Mitigation by design

Within this question we define baselines in order to mitigate attacks by defining guidelines for using the protocol. We analyse the effectiveness of the guidelines and try to elaborate the general achievement level of the protocol by looking at the criteria defined in SQ1.

This question will mainly be answered in part III



Part I.

Methodes

In this part of the thesis we collect requirements, definitions, methods, and existing research relevant to the topic of this thesis. We start with collecting requirements for the protocol.

Having the requirements we collect numerous existing technologies on research and implementation level. Each of the technologies is quickly categorized and either further studied or rejected naming the reasons for rejection.

The list of technologies and research collected is big in this chapter. All relevant technologies either widely adopted or thoroughly researched should be included in this chapter. All Technologies and research are categorized.

Technologies are always referenced through their respective standard. If applicable multiple standards may be part of the analysis. A very quick introduction into the protocol is given and then analysed for suitability for a specific problem addressed in this work.

Research are referenced by a paper. If applicable multiple related researches and papers are collected together into a bigger picture and then analysed for suitability concerning specific problem. If related to a standard technology links to that technology are provided where relevant.

The chapters have been chosen by possible relevance. In chapter 3 we collect the requirements for the protocol.

Based on findings in this section we collect in chapter 4.

In chapter 5 we collect relevant research to the topic.

In chapter 6 we summarize the identified methods which will be used. We explain choices and give a rough outline of the protocol baselines.

3. Requirements for an anonymising Protocol

In the following sections we try to elaborate the main characteristics of the anonymising protocol.

The main 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)[105].

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 in print, in the form of art, or through any other media of his choice.

We imply that not all participants in the internet share this value. As of September 1st 2016 Countries such as China (signatory), Cuba (signatory), Quatar, Saudi Arabia, Singapore, United Arab Emirates, or Myanmar did not ratify the ICCPR. Other countries such as 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 in the internet as a lot of countries explicitly supersede them.

We always have to keep in mind that we have no control over the flow of data packets in the internet. Packets may pass though any point of the world. It is not even possible to detect what way has a packet taken. The common network diagnostic tool `tracert` respectively `traceroute` tries to figure that out. But neither can a route of a packet beeing sent forseen nor can it be measured while or after sending. This is due to the fact that all routers along the way only decide for the next hop of a packet.

As an example of the problems analysing a packet route we may look at `traceroute`. The manpage of traceroute of Linux tells us that traceroute uses UDP, TCP, or ICMP packets with a short TTL and analyses the IP of the peer sending an TIME_EXCEEDED (message of the

ICMP protocol). This information is then collected and shown as a route. This route may be completely wrong. Some of the possible cases are described in the manpage.

The output of `traceroute` is therefore not a reliable indication of route. Since routes do not have to be static and may be changed or even be alternating the output represents in the best case a snapshot of the current routing situation. We cannot be sure that data packets we are sending are passing only through countries accepting the ICCPR to the full extend.

```
$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 3.1.: A traceroute to the host www.ietf.org

3.1. Anonymizing and Unlinking

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

As a result anonymisation of a sender or a receiver is not simple. If messages are being sent through a relay at least the original sender might be concealed (Sender anonymity). By combining it with encryption we may even achieve a very simple form of sender and receiver pseudonymity. If cascading more relay like infrastructures and combining it with cryptography we might even

achieve sender and receiver anonymity. This approach has however several downsides (see 5.9.4.2 and 5.9.1 for details) and is easily attackable.

3.2. Censorship Resistant

In our scenario in 3 we defined the adversary as someone with superior access to the internet and its infrastructure. Such an adversary might attack a message flow in several ways:

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

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

In order to be able to withstand an adversary outlined above the protocol needs certain attributes. The message content needs to be unidentifiable by attributes or content. Whereas “Attributes” include meta information such as frequency, timing, message size, sender, protocol, ports, or recipient (list not conclusive).

3.3. Controllable trust

If we want to control trust we have to conclude that

1. trust in an infrastructure is given due to the fact that it is under full control of either the sender or the recipient.
2. trust in an infrastructure may not be necessary as the infrastructure is ideally unable to misuse data passed through it.

In this thesis we work with both cases. We will however never trust a third party apart from sender and recipient (no even their providers of infrastructures).

3.4. Reliable

Any message sending protocol needs to be reliable in its functionality. If means of message transport are unreliable users tend to use different means for communication.

3.5. Diagnosable

Reliability is somehow linked to transparent behaviour. This due to the fact that if something is generating a constant behaviour but we are unable to determine the reason for it (i.e. if we are expecting a different behaviour) we normally assume a malfunction. Therefore “Reliable” means not only stable by its behaviour. It means also diagnosable. User perception will not be “Reliable” if he is not able to determine causes for differences in observed and expected behaviour.

3.6. Available

Availability has two meanings in this context which do differ. A technology is available if...

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

The first meaning tells us that a protocol must run independently from infrastructure the user has commonly ready.

The second meaning tells us that messages must always be capable of flowing from the sender to the recipient. As infrastructure may fail at any time the protocol must offer the possibilities to send messages through alternate routes. This sounds easy and many protocols implement such redundancies already. However, taking into account that sender and recipient is not known to a routing node this is a goal hard to achieve.

3.7. Rough draft of Protocol Layers

In order to fulfil the criteria given above we define a very rough idea of the protocol and its layers. The rough overview is given in figure 3.2. The protocol should work on the base of onion routing. Unlike Tor (see 5.10.8) it should not rely on central infrastructures. It furthermore should not be limited to onionize messages. It should be capable of splitting and reassemble messages at any layer

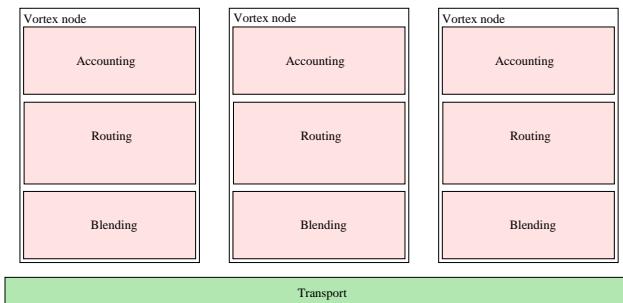


Figure 3.2.: A first rough overview for the protocol

The protocol itself should send messages through a well known transport protocol ("Transport") which will be used for our messages.

The messages will be hidden within regular messages already using that transport layer. In an ideal implementation messages sent by our protocol should be indistinguishable from regular messages (by computers and humans). In order to achieve this particular task we introduce a "Hiding / Blending" layer. This layer is normally specific to the underlying transport layer and may vary.

The "Routing" layer is the layer that receives and sends messages. It parses only the core protocol and the data processed here is completely independent from the underlying transport layer.

The "Assembly / Accounting" layer disassembles the received messages into its parts. It then recomposes and stores the messages for further routing. In order to avoid spamming over this media some kind of accounting will be involved.

The messages passed through such a protocol stack are onionised to offer anonymity from evil nodes. Furthermore a node should be unable to tell whether a message was intended for a specific node or not (so all nodes including sender and recipient should handle identically).

In order to mitigate size based attacks (an onionised packet loses permanently data on its way) we introduce operations which may be applied to a valid message as well as to decoy traffic. The operations should be able to increase and decrease in size without being able to tell if this message is destined to a target or not. Operations not carried out due to missing data might be either intentionally or a malfunction.

The defined message operations are as follows:

- Split a message block into two parts of variable size.
This operation may be used to fan out decoy traffic, to cut off decoy data, or to send multiple parts of a message through different routing nodes.
- Join two data blocks to a bigger block using an xor operation.
This operation may be used to minimise decoy traffic or to rejoin data which has been split earlier.
- Fork off xor'ed data from a data block by applying an xor operation to a received block with a block of random data.
With this operation we may create decoy traffic, or split a message into two parts.
- Encrypt a block with a given key.
- Decrypt a block with a given key.

To avoid making this system attractive to UBE an accounting system is introduced. The system works as follows:

- Routing nodes may offer their services at a cost. These "costs" are typically paid by solving crypto puzzles.
- The following operations may be cost effective:
 - Getting a discardable identity on a node (prerequisite for accounting).
 - Allowance to route a number of messages through the node (cost per outgoing message) in a specified interval. This allowance is always linked to a discardable identity.
 - Allowance to route a specific size of bytes through the node (cost per outgoing byte) in a specified time interval. This allowance is always linked to a discardable identity.

- Offer information about the known routing network by the node.
- Offer information about the identities current quota.

All quotas require accounting. In order to minimise accounting all these quotas must be assigned a time interval in which they are effective. This guarantees that quota data is not growing indefinitely. It is up to the node to decide what time duration is still acceptable.

In order to limit the possibility to Denial of Service (DoS) a node by overloading it the following precautions are taken:

- The message is built in such a way that message building is far more complex than message routing.
- Messages may be decrypted in parts to minimize the amount of work required to decide whether the full processing of the message will be done or the message is discarded anyway.
- The usage of inefficient operations (e.g. asymmetric encryption) is minimized.
- The server may limit specific operations.

3.8. Rough Draft of Messages

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

- Header block

This block contains an ephemeral identity of the sender on the message processing host. It allows the host to decide whether he is willing to process the rest of the message or not. It is important to know that the identity block contains a symmetric key for decryption of the main block and a secret repeated in the main block. This prevents that a malicious node may exchange the main block.

- Main block

This block contains all information regarding routing and processing data. It furthermore contains the payload data which may or may not contain messages.

- Routing Blocks

This block contains information of which data block should be sent to what re-

cipient. It furthermore may contain instruction for processing the data blocks and routing/reply blocks for subsequent processing. Another important thing to note is that routing blocks and payload data may arrive on two different paths to the target node.

- Routing Log Block

This Block contains information about the already occurred routing in onionised form. It may be regarded as the onionised pendant of received header in an SMTP header.

- Reply Block

This Block contains a Routing block which may be used to contact the original sender of a message (e.g. for an error message)

- payload Block

These Blocks form the payload data. It might contain a message, parts of a message or decoy material.

The message is picked up from the transport layer and extracted in the blending layer. Processing is then split. First the Assembly/accounting layer is extracting the identity block and authorises further processing. The message building instructions are put into a identity specific list (with the expiration dates of the message). and the payload blocks are stored (again with the expiration date).

After having prepared the message in such a way the routing layer gets the routing blocks. Every routing block is then processed after a final approval by the accounting layer (he is in charge to keep the quotas for this identity in sync). The routing layer assembles the new messages according to the (no authorized) instructions in the routing block and sends it to the next hop where the processing restarts with the new instructions.

4. Existing Transport Layer Protocols

In this chapter we have a look at various available transport layer protocol. The main goal is to identify strong candidates as a transport layer. Main focus lies on the following criterias:

- Widely adopted (Ct1)

The more widely adopted and used a protocol is the harder it is for an adversary to monitor (due to the sheer mass), filter or block the protocol (censorship resistance).

- Reliable (Ct2)

Message transport between peers should be reliable. As messages may arrive anytime from everywhere we don not have means to synchronize the peer partners on a higher level without investing a considerable effort. In order to avoid this effort we do look for inherently reliable protocols.

- Symmetrical built (Ct3)

The transport layer should be built on a peer to peer base. All servers implement a generic routing which requires no prior knowledge of possible targets. This criteria neglects centralised infrastructures.

The reply onto 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 (as specified in [14]) is using the same ports and default behaviour. Unlike HTTP/0.9-HTTP/1.1 it is not a clear text but a binary protocol. Headers and bodies of messages are sent as binaries.



The protocol does definitely satisfy the first two main criteria (Ct1: Widely Adopted and Ct2: Reliable).

The main disadvantage in terms as message transport protocol is that this protocol is not symmetrically. This means that a server is always just “serving requests” and not sending actively information to peers. This Request-Reply violates criteria (Ct1: Symmetrically built) and makes the protocol not a primary choice for message transport.

4.1. HTTP

The HTTP protocol allows message transfer from and to a server and is specified in RFC2616 [75]. It is not suitable as a communication protocol for messages due to the lack of notifications. There are some extensions which would allow such communications (such as WebDAV) but in general even those are not suitable as they require a continuous connection to the server in order 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 which would be suitable for transferring messages to the server.

4.2. FTP

FTP is defined in RFC959[82]. This Protocol is intended for authenticated file transfer only. There is an account available for general access (“anonymous”). This account does for security reasons normally not offer upload rights.

It is possible to use FTP as message transfer endpoint. The configuration would work as follows: “anonymous” has upload rights only. It is unable to download or list a directory. A node may upload a message with a random name. If there is a collision the node retries with another random name.

The blending layer picks messages up using an authenticated user.

This has multiple downsides. At first, handling FTP that way is very uncommon and usually requires an own dedicated infrastructure. Secondly passwords are within FTP always in plain text.

This is considered as a very bad practice nowadays. 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 transport layer protocol.

Like HTTP a main disadvantage of FTP in terms as a message transport protocol is that this protocol is not symmetrically. This means that a server is always just “serving requests” and not sending actively information 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).

time of writing this document was 3.1.1 [7].

The protocol runs by default on the two ports 1883 and 8883 and may be encrypted with TLS. MQTT is a publish/subscribe based message passing protocol which is mainly targeted to m2m communication. This Protocol requires the receiving party to be subscribed in 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 definitely satisfy the second criteria (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 decentralistic design the protocol fails as well (Ct3: Symmetrically built).

4.3. 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. This makes it not suitable as transport layer as it would leave messages open to anyone. 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 in the internet (it is quite commonly used in lans for booting over a network connection). This violates Criteria one (Ct1: Widely Adopted). TFTP is not symmetrically. This means that a server is always just “serving requests” and not sending actively information 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.

4.5. Advanced Message Queuing Protocol

The Advanced Message Queuing Protocol (AMQP) was originally initiated by numerous exponents based mainly in finance related industries. The AMQP-Protocol is either used for communication between two message brokers, or between a message broker and a client[6].

It is designed to be interoperable, stable, reliable and safe. It supports either SASL or TLS secured communication. The usage 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[6].

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

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

4.4. MQTT

MQTT is an ISO standard (ISO/IEC PRF 20922:2016) and was formerly called MQ Telemetry Transport. The current standard as the

4.6. 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[20]. It is Defined as 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 criteria (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 as 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 actively information to peers. This Request-Reply violates criteria (Ct3: Symmetrically built) and makes the protocol not a primary choice for message transport.

files between endpoints [102].

It has generic routing capabilities spanning between known and unknown servers.

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



4.9. SMTP

The SMTP protocol is currently specified in [61]. It specifies a method to deliver reliably asynchronous mail objects thru 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 contains 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. Exact outline of the header is specified in [86] and is separated with a blank line from the body.

It [61] furthermore introduces a simplistic model for smtp message based communication. A more comprehensive model is introduced in the section SMTP. As the proposed model is not sufficient for a comprehensive end-to-end analysis.

Traditionally the message itself is mime encoded. The MIME messages are mainly specified in [43], and [44]. 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 in the internet (Ct1: Widely Adopted)

4.7. Web Application Messaging Protocol

WAMP is a websockets based protocol destined to enable M2M communication. Like MQTT it is publish/subscribe oriented. Unlike MQTT it allows remote procedure calls (RPC).

The WAMP protocol is not widely adopted (Ct1: Widely Adopted) but it is definitely 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 behaviour is therefore not available but could be built in relatively easy.

4.8. XMPP (jabber)

XMPP (originally named Jabber) is a synchronous message protocol used in the internet. It is specified in the documents RFC6120[93], RFC6121[93], RFC3922[92], and RFC3923[91]. The protocol is a very advanced chat protocol featuring numeros levels of security including end-to-end signing and object encryption[91]. There is also a stream initiation extension for transferring

and it would be devastating for 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 throughout message design making it relatively easy to diagnose problems (Ct2: Reliable). All SMTP servers are normally capable of routing and receiving messages. Messages going over several servers are common (Ct3: Symmetrically built) so the third criteria may be considered as fulfilled as well.

SMTP is considered a strong candidate as transport layer.

4.10. SMS

SMS capability was introduced in the SS7 protocol. This protocol allows the message transfer of messages not bigger than 144 character. 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.

Secondly the protocol is not widely adopted within the internet domain. There are gateways providing bridging functionalities to the SMS service. However – the protocol itself is insignificant in the internet itself.

4.11. MMS

The Multimedia Messaging Service (MMS) is maintained by 3GPP (3rd Generation Partnership Project). This protocol is manly a mobile protocol based on telephone networks. This protocol is just like the SMS protocol accessible through the internet by using gateways but not directly usable within the internet.

4.12. Roundup for Transport Protocols

In table 4.1 we sum up all previously analysed protocols. We use “✓” for a fulfilled criteria, “~” for a partially fulfilled criteria, and “✗” for a not fulfilled criteria. This overview shows in compact form which protocols have been identified as

| Protocol \ Criteria | Ct1: Widely adopted | Ct2: Reliable | Ct3: Symmetrically built |
|---------------------|---------------------|---------------|--------------------------|
| HTTP | ✓ | ✓ | ✗ |
| FTP | ✓ | ✓ | ✗ |
| TFTP | ✗ | ✗ | ✗ |
| MQTT | ~ | ✓ | ✗ |
| AMQP | ~ | ✓ | ✗ |
| CoAP | ~ | ~ | ✗ |
| WAMP | ✗ | ✓ | ~ |
| XMPP | ✓ | ✓ | ✓ |
| SMTP | ✓ | ✓ | ✓ |
| SMS ¹ | | | |
| MMS | ✗ | ✓ | ✗ |

Table 4.1.: comparison of protocols in terms of the suitability criteria as transport layer

strong candidates for use as a transport layer in terms of an anonymising protocol.

This table shows that strong identified candidates are SMTP (being already a message sending protocol on asynchronous base) and XMPP (a real time chat protocol able to attach files. This protocol features furthermore end-to-end encryption and signing). Both have the advantages that they are really wide adopted in the internet and do support additionally advanced content (such as alternatives or attachments).

¹omitted due to message size being too small

5. Existing Research and Implementations on the Topic

5.1. Anonymity

As Anonymity we take the definition as specified in [81].

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

Whereas the anonymity set is defined as the set of all possible subjects.

Especially the anonymity of a subject from an attacker's is very important to this paper.

5.1.1. *k*-Anonymity

k-anonymity is a term introduced in [3]. This work claims that no one might be held responsible for an action if the action itself can only be identified as an action which has been taken by one unidentifiable entity out of *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 key factor. One of the criteria might be the legal requirements of a jurisdiction. Depending on the jurisdiction it is normally not possible prosecute someone if an action is not directly coupled to one person. Another criteria 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.

¹footnotes omitted in quote

5.1.2. ℓ -Diversity

In [67] 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 special entity can be filtered from the *k*-anonymity set. In other words if an anonymity set is to tightly specified a single additional background information might be sufficient to identify a specific entity in an anonymity set.

While 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 the point made in this work is definitely right and should be taken into account.

Their approach is to introduce an amount of invisible diversity into *k*-anonymous sets so that simple background knowledge is no longer sufficient to isolate a single member.

5.1.3. *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[77] 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.

5.2. 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.

5.3. Pseudonymity

As Pseudonymity we take the definition as specified in [81].

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 identifier instead of one of its real names.²

5.4. Undetectability

As undetectability we take the definition as specified in [81].

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

5.5. Unobservability

As unobservability we take the definition as specified in [81].

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.

This part is very important. As we are heading for a censorship resistant solution unobservability is a key. As mentioned in this paper unobservability raises the bar of required attributes again (⇒ reads "implies"):

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

So this means that we have to use an undetectable data channel in order to achieve censorship resistance.

5.5.1. Ephemeral Identity

In this work we use accounting on various levels. While we are dealing with anonymity accounting has still to be linked to some kind of identity for this reason we are introducing a term called "ephemeral identity".

A Ephemeral identity is a temporary identity which is defined by the following attributes:

- It is an artificial identity
- It is only used for a short timespan
- It is not linkable to another identity

The key in this definition is the last point is crucial and at the same time hard to achieve.

5.6. Single Use Reply Blocks and Multi Use Reply Blocks

The use of single use reply blocks were first introduced by Chaum in [23]. A routing block in general is a structure allowing to send a message to someone without knowing the targets true address. It might be differentiated into "Single Use Reply Bolcks" (SURBs) which may be used once and "Multi Use Reply Blocks" (MURBs) which may be used a limited number of times.

²footnotes omitted in quote

The concept is that we have in our case a routing block which might be used up to n times ($0 < n < 127$). The number has been chosen for practical reasons. It is easily representable in a byte integer (signed or unsigned) on any system. It is big enough to support human communication in a sensible way 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.

5.7. Censorship

As 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".

Which 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 subsystem this means

A systematic suppression, modification, or banning of data in the internet by either removal, or modification of the data, or systematic influencing of entities involved in the processing (e.g. by writing, routing, storing or reading) of this data.

5.7.1. Censorship Resistant

A censorship resistant system is a system which allows the users of the system and the data itself to be unaffected from censorship. Please note that this does not deny the possibility of censorship per se. It still exists outside the system. But 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 means of suppressing real world entities participating in the system.

5.7.2. Parrot Circumvention

In [55] Housmansadr, Brubaker, and Shmatikov express that it is easy for a human to determine decoy traffic as the content is easily identifiable as automated content. While this is absolutely true there is a possibility here to generate "human like" data traffic to a certain extent. In our design this is the job covered by the blending layer.

5.7.3. Censorship Circumvention

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

- Hide data.
- Copy data to a vast amount of places in order to improve the lifespan of data.

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

5.7.3.1. Covert Channel and Channel Exploitations

The original term of covert channels was defined by Lampson[63] 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 in this specific channel is

not a usage of covert channel. This method is commonly referenced as channel exploitation. In the following section we analyse technologies for suitability in a P2P network construct as data channel.

5.7.3.2. Spread Spectrum

[106] 

5.7.3.3. Steganography

According to [110] steganography is the art of hiding information in carrier data. The carrier data is commonly called container. The modified container containing the hidden information is commonly called steganogram.

[99] 



f5 

YASS 

5.7.3.4. Timing Channels

Timing channels are a specialised form of covert channels. In timing channels not the information itself is written into the channel but the usage of the channel is encoded in such a way that it is capable to reflect the data being transferred.



5.8. Cryptography

Whenever dealing with hiding data and maintaining 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 did not collect a lot of information. We focussed on algorithms either well researched and implemented or research which seem very valuable when putting this work into place.

5.8.1. Symmetric Encryption

Symmetric encryption in this paper assumes always that

$$D^{K_a}(E^{K_a}(\mathbf{M})) = \mathbf{M} \quad (5.1)$$

For a key $K_b \neq K_a$ this means

$$D^{K_a}(E^{K_b}(\mathbf{M})) \neq \mathbf{M} \quad (5.2)$$

$$D^{K_b}(E^{K_a}(\mathbf{M})) \neq \mathbf{M} \quad (5.3)$$

5.8.1.1. AES

AES was announced by NIST in 2001 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 (2016) unbroken. It has been weakened in the analysis described in [100] which reduces the complexity by roughly one to two bits.

5.8.1.2. Camellia

The camellia algorithm is described in [68]. The key sizes are 128, 192, and 256. Camellia is a Feinstel cypher with 18 to 24 rounds depending on the key size. Up until today the cypher is not known to be broken.

5.8.2. Asymmetric Encryption

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

$$D^{K_a^{-1}}(E^{K_a^1}(\mathbf{M})) = \mathbf{M} \quad (5.4)$$

$$D^{K_a^1}(E^{K_a^{-1}}(\mathbf{M})) = \mathbf{M} \quad (5.5)$$

It is important that

$$D^{K_a^{-1}}(E^{K_a^{-1}}(\mathbf{M})) \neq \mathbf{M} \quad (5.6)$$

$$D^{K_a^1}(E^{K_a^1}(\mathbf{M})) \neq \mathbf{M} \quad (5.7)$$

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

$$D^{K_b^{-1}}(E^{K_a^1}(M)) \neq M \quad (5.8)$$

$$D^{K_b^1}(E^{K_a^1}(M)) \neq M \quad (5.9)$$

$$D^{K_b^{-1}}(E^{K_a^{-1}}(M)) \neq M \quad (5.10)$$

$$D^{K_b^1}(E^{K_a^{-1}}(M)) \neq M \quad (5.11)$$

5.8.5. Key Sizes

The question of key sizes is a 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 is 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 Lenstra in [64] an asymmetric key size of 2644 Bits respectively a 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 [39] data classified up to “top secret” should be secured asymmetric encryption based on factoring modulus (no proposed encryption standard). For symmetric encryption they recommend 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 Lenstra do not explicitly take quantum computers into account we therefore follow the recommendations of ENCRYPT II.

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

5.8.2.2. Elliptic Curve Cryptography

The elliptic curves were independently suggested by [71] and [62] in 1986. Elliptic curve Cryptography started to be widely deployed in the public space about 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.

5.8.3. Homomorphic encryption



[47]

5.8.6. Cypher Mode

The cypher mode defines how multiple blocks encrypted with the same key are handled. Main characteristics of cypher suites are:

- Parallelisable

Can multiple parts of a plaintext be encrypted simultaneously This feature is important for multi CPU and multi core systems

5.8.4. Deniable Encryption and Deniable Steganography



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 cypher text.

- Initialisation vector

An initialisation vector has downsides and advantages. On the downsides is the fact that an initialisation vector must be shared with the message or prior distributing it. It is important to understand that the initialisation vector itself is normally not treated as a secret. It is not part of the key.

- Authentication



5.8.6.4. EAX:

EAX has been broken in 2012[72] and is therefore rejected for our use.

5.8.6.5. CFB (Cypher Feedback):

CFB is specified in [34] and works exactly as CBC with the difference that the plain text is xor'ed and the initialisation vector respectively the preceding cypher result is encrypted. CFB does not support parallel encryption as the ciphertext input from the preceding operation is required for a round, it does however allow parallel decryption and random access.

5.8.6.1. ECB (Electronic Code Book):

This is the most basic operation. Each block is encrypted on its own. This results in a big flaw. blocks containing the same data will always transform to the same byte code. This makes it possible to see some structures of the plain text when looking at the cypher text. This solution allows parallelisation of encryption, decryption, and random access while decrypting.

5.8.6.2. CBC (Cypher Block Chaining):

CBC extends the encryption by xor'ing an initialisation 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 parallelisation of encryption, but decryption may be parallelized and random access is possible. As another downside CBC requires a shared initialisation vector.

5.8.6.6. OFB:

OFB is specified in [34] and works exactly as CFB except for the fact that not the cypher text 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.

5.8.6.7. OCB (Offset Codebook Mode):

This mode was first proposed in [89] and later specified in [90]. 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 specialised function $H_{OCB}(M)$. OCB is fully parallelizable due to its internal structure. All blocks except the first and the last may be encrypted or decrypted in parallel.



5.8.6.3. PCBC (Propagation Cypher Block Chaining):

PCBC extends the encryption by xor'ing an initialisation vector into the first block before encrypting. For all subsequent blocks the ciphertext result of the preceding block xor'ed with its plain text is taken as xor input. This solution does not allow parallel decryption or encryption, nor does it allow random access. As another downside PCBC requires a shared initialisation vector.

5.8.6.8. CTR:

CTR is specified in [66] 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 allows parallel decryption and encryption as well as random access.

5.8.6.9. CCM:

Counter with CBC-MAC (CCM) is specified in [109]. 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 of 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 therefore shares the same properties like CTR.

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

5.8.7. 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. This mechanism is called padding.

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

5.8.7.1. PKCS1:



5.8.7.2. PKCS7:

This padding is the standard used in symmetric encryption up to 256 bits key length. The free bytes in the last cypher 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.

5.8.7.3. OAEP with SHA and MGF1 padding:

This padding is closely related to PKCS1 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 three paddings. The allowed sha sizes match the allowed hash sizes chosen above. It is important to note that padding costs space at the end of a stream. Since we are using always 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 is normally not a problem for RSA as there are keys 1024+ Bits required it is a considerable problem for ECC algorithms as there are much shorter keys required to achieve an equivalent strength compared to RSA.

5.8.6.10. GCM (Galois Counter Mode):

GCM has been defined in [69], and is related to CTR but has some major differences. The nonce is not used (just the counter starting with value 1). In order to authenticate the encryption an authentication token $auth$ is hashed with H_{GFmult} and then xor'ed with the first cypher 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:

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

The mode has been analysed security wise in 2004 and showed no weaknesses in the analysed fields [70].

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

| Criteria Mode | Requires IV | binary proof | parallelisable |
|------------------|-----------------|--------------|----------------|
| GCM | ✓ | ✓ | ✗ |
| CTR | ✓ | ✓ | ✓ |
| CCM | ✓ | ✓ | ✗ |
| CFB | ✓ | ✓ | ✓ |
| OCB | ✗ includes auth | ✓ | ✗ |
| OFB | ✓ | ✓ | ✗ |
| PCBC | ✓ | ✓ | ✗ |
| CBC | ✓ | ✓ | ✗ |
| ECB | ✗ | ✓ | ✓ |

Table 5.1.: comparison of encryption modes in terms of the suitability criteria as transport layer



5.9. Routing

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

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

5.9.1. Mixing

Mixes have been first introduced by “Untraceable Electronic Mail, Return, Addresses, and Digital Pseudonyms”[23] in 1981. 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 3 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 analysing 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 a content or an attribute) we then may be able to follow the message.

In 2003 Rennhard and Plattner analyzed the suit-

ability for mixes as a 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.

5.9.2. Onion Routing

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

While tagging attacks are far harder (if we exclude side channel attacks to break sender anonymity) the traditional attacks on mixes are still possible. So when adversaries are operating entry and/or exit nodes it is very easy for them to match the respective traffic.

One very well known of onion routing networks is Tor (<https://www.torproject.org>). For more information about tor see section 5.10.8.



5.9.3. Crowds

Crowds is a network which 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 Johndo 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 do 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 iondos.



5.9.4. Mimic routes

Mimics are a set of statical mixes which maintain a constant message flow between the static routes. If legitimate traffic arrives the pseudo traffic is replaced by the 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 sender and receiver operate a mixer by themselves the system would be no longer 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) new mimic load.

5.9.4.1. DC Networks

DC networks are based on the work “The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability” by Chaum[24]. In this work Chaum 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[49][107]. But no one succeeded without introducing significant downsides on the privacy side.

5.9.4.2. Anonymous Remailer

Remailers have been in use for quite some time. There are several classes of remailers and all of them are somehow related to Mixnets. There

are “types” of remailers defined. Although these “types” offer some kind of 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 5.10.1

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 I Remailers see 5.10.3.

Mixmaster remailers are very similar to Cypherpunk remailers. Unlike them Mixmaster remailers hide the messages not in an own protocol but use SMTP instead for it. 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 II remailers see 5.10.4.

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

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

For a more in depth discussion of Mixminion remailers see 5.10.5.



5.10. System Implementations

The following sections emphasize on implementations of anonymising (and related) protocols regardless of their usage in the domain of messaging. It is a list of system classes or their specific implementations together with a short analysis of strength and weaknesses. Wherever possible we try to refer to original sources. This is however not always possible since some of these systems are no longer in use.

If a system shows strong similarities in parts then we emphasize on this parts and analyse the findings.

5.10.1. Pseudonymous Remailer

The basic idea of remailers were discussed in [23]. The most well known remailer was probably anon.penit.fi which operated from 1993 to 1996.

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

As the example shows this kind of remailer is easily attackable by an authority. Since the remailer knows tuples of pseudonyms and their respective real identity it may be forced to reveal true identities. Furthermore message may be monitored at the server or on its way and then due to the content a matching or even tagging is possible.

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

5.10.2. Babel

Babel was an academic system defined in a paper by Gülcü and Tsudik in 1996[53].

5.10.3. Cypherpunk-Remailer

5.10.4. Mixmaster-Remailer



5.10.5. Mixminion-Remailer



5.10.6. Crowds

Crowds is an implementation of the crowds protocol.



5.10.7. Tarzan



5.10.8. Tor

Tor is one of the most common onion router networks these days. It is specified in [31]. It might be considered one of the most advanced networks since a lot of research has been done here.

In short 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. This path is linked to the temporary identity. This identity (and the path connected to it) should be changed at least once a day.

There is 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 [78, 12, 13, 18, 19, 30, 38] and some have even been exploited actively. In the best case the people discovering the attacks did propose mitigation to the attack and took care that 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) seems to be accepted as one might

argue that users of Tor are not regarded bad in general. 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. This is why tor nodes may be listed as “non exit nodes”

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

Attacking anonymisation has been done by 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 huge attack of this kind was published in 2014 and has been published on the tor website (relay early traffic confirmation attack). This has been possible because tor is a low latency network. Another attack is to identify routes through tor by statistically analyse the traffic density in the network between nodes. More theoretical attacks focus on the possibility of controlling the directory servers to guarantee that a entity may be deanonymised because it is using compromised routers.

Generally the effectiveness of monitoring of single or multiple networks is disputed. According to a study by Johnson et al. in 2013[58] 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 Tor users. However since these papers referenced to “manual analysis” the statement may be disputed when looking at automated attacks as well.



5.10.9. I^2P

The name I^2P is an derived from “Invisible Internet Project” according to geti2p.net. The system itself is comparable to Tor for its capabilities. Mayor differences are:

- P2P based
- Packet switched routing (tor is “circuit switched”)
- Different forward and backward routes

(called tunnels)

- Works pseudonymously
- Supports TCP and UDP

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

In 2011 Herrmann and Grothoff presented in [54] an attack. As I^2P s security model is chosen based on IP adresses 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 may be influenced by an adversary enabling him to recover data on a statistical base.

5.10.10. Freenet

Freenet was originally designed to be a fully distributed data store[25]. 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 main goal of Freenet is to decouple authorship from a particular document. It furthermore provides a fault tolerant storage which improves caching of a document if requested more often.

Exactly as I^2P it is not analysed 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 insert and retrieve data, query the network status and managing freenet nodes directly connected to an own node. FCPv2 operates on port 9481 and may be easily blocked 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).

5.10.11. Herbivore

Herbivore is a network protocol designed by Goel et al. in [48]. It is based on the dining cryptographers paper of Chaum. 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”.

5.10.15. Gnutella2

Despite its name Gnutella2 is not the next generation of Gnutella. It was a fork in 2002 from the original Gnutella and has been developed in a different direction. The specification may be found on <http://g2.doxu.org>.



5.10.12. Dissent

Dissent is defined in [28]. It is an anonymity network based on DC-nets. These DC-nets are formed by a set of servers. 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.

At the time of writing no Dissent client or server could be found in the internet. The last paper[112] found was dated 2012.

5.10.16. Hordes

Hordes was specified in [65]. Hordes uses the abilities of handling 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 true for a local observer. Unfortunately it is not sufficient assuming an adversary as defined in this paper.

5.10.13. \mathcal{P}^5

The Peer-to-Peer Personal Privacy Protocol is defined in [94]. It provides sender-, receiver- and sender-receiver anonymity. Unlike many other protocols. According to the project page of \mathcal{P}^5 there is only a simulator available for the protocol.

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

5.10.17. Salsa

[76]

Related: [74]



5.10.18. AP3

[73]



5.10.14. Gnutella

Gnutella is not a protocol to the anonymity world in special. It is a file sharing protocol on a Peer to peer base. This is the most interesting aspect of gnutella in this context. 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/>

extension: [8]

5.10.19. Hydra-Onion

[56]



5.10.20. DUO-Onion

[56]



5.10.21. Cashmere

Cashmere is specified in [113]. It defines a protocol for the use of chaum mixes. Unlike most of the protocols the chaum mixes in cashmere are virtual. They are represented by so called relay groups. Each mix in the relay group may be used as an equivalent mix to all other mixes in the same group.

This 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.

5.10.22. SMTP

As SMTP is our transport prototype we focus in depth onto this topic.

Todays mail transport is mostly done via SMTP protocol as specified in [61]. This protocol has proven to be stable and reliable. Most of the messages are passed from a MUA to a SMTP relay of a provider. From there the message is directly sent to the SMTP server of the recipient and from there to a 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 (where as the later 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 I used a consistent model which includes all components (server and clients). The figure 5.1 shows all involved parties of a typical Mail routing. It is important to understand that Mail routing remains the same regardless of the used client. However – 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 (derfrefStorage), agent (derfrefAgent) and service

(derfrefService). Storages are endpoint storages storing mails. 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 which is responsible for a specific task or task sets.

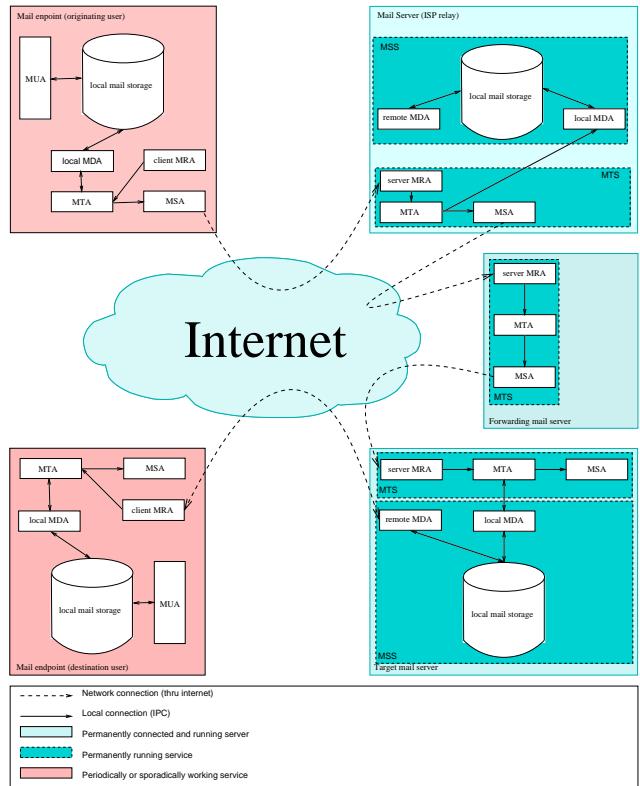


Figure 5.1.: Mail Agents

In the following paragraphs (for definitions) the term “Mail” is used synonymously to the term “Message”. The reason why “Mail” has been chosen over “Messages” that a lot of terms do already exist in standard documents. In these documents the term mail is commonly used.

Mails are typically initiated by a Mail User Agent (MUA). A MUA accesses a local mail 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 mails 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 a local mail storage is the Mail Delivery Agent (MDA). An MDA is responsible for storing and fetching mails from the local mail storage. Mails destined for other accounts than the current

one are forwarded to the MTA. Mails destined to a User are persistently stored in the local mailstorage. It is important to understand that a mailstorage not necessarily reflects a simple mailbox. It may as well represent multiple mailboxes (eg. a rich client serving multiple IMAP accounts) or a combined view of multiple accounts (eg. 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 a mail server the local MDA is part of the local Mailstore (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.

A MTS consists generally 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 mails. And a Mail Sending Agent (MSA) which is responsible for transmitting mails reliably to another Server MRA (usually sent via SMTP).

A MSS consists out of a 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 kind of MDAs. Local MDAs offer possibilities to access the store via efficient (non network based) mechanisms (eg IPC or named sockets). This is usually done with a stripped down protocol (eg. LMTP). For remote agents there a publicly – network based – agent available. Common Protocols for this Remote MDA include POP, IMAP, or MS-OXMAPIHTTP.

5.10.23. Mail Endpoints



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 specialities.

- Portnumber is 587 (specified in [46]). Although port numbers 25 and 465 are valid and do have usually 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 (eg. username/password or certificate) or data identifying the sending system (eg. IP address)[46]. Failure in doing authentication may result in this port being misused as a sender for UBE.

Mail User Agents (MUA) are the terminal endpoint of a mail 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 a 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.

5.10.23.1. Fat clients

The majority of mail clients are fat clients. These clients score over the more centralistic organized web clients in the way that they may offer mail availability even if an internet connection is not available (through a client specific local mail storage). They furthermore provide the possibility to collect mails from multiple sources and store

them in the local storage. Unlike Mail servers, clients are assumed to be not always online. In fact they may be offline most of the time. To guarantee the availability of a certain email address a responsible mail server for a specific address collects all mails (this is done by the MSS) and provides a consolidated view onto the database when a client connects through a local or remote MDA.

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

Fat clients are commonly used on mobile devices. According to [22] in Aug 2012 the most common fat email client was Apple Mail client on iOS devices (35.6%), followed by Outlook (20.14%), and Apple Mail (11%). *Email Client Market Share*[36] as a more recent source lists in February 2014 iOS devices with 37%, followed by Outlook (13%), and Google Android (9%).

5.10.23.2. Server located clients

server located clients build an absolute minority. This kind of clients have been used mainly in the days of centralized hosts. An example for a Server Located Client is the Unix command "mail". This client reads a mail storage from a file in the users home directory.

5.10.23.3. Web clients

Web clients are these days a common alternative to fat clients. Most big provider companies use their own proprietary web client. According to [36] 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.

5.10.24. Interfaces of Mail Endpoints

There are two interfaces



5.10.25. S/MIME

S/MIME is an extension to the MIME standard. The MIME standard allows in simple text oriented mails an alternate representation of the same content (e.g. as text and as html), or it allows to split a message into multiple parts which 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 [84] 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 important to know, that this method leaks significant parts 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 are 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.

5.10.26. PGP/MIME

Exactly as S/MIME PGP[83] 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).

A good thing to learn from PGP is that there are peer based approaches 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.



5.10.27. XMMP



5.11. Proof of work

Proof of work is one possibility to limit resource exhaustion on locally controlled resources. Traditional proofs are either CPU- or memory-bound.

This means that they have to proof that they assigned a certain amount of their own CPU time or memory bandwidth to a problem posed by the receiving node.

In [5] is a good summary of proposed proof of work methods. Although they are used there for the defence of DDoS-attacks (Distributed Denial of Service) they might as well be used for defence against misuse for UBE and (D)DoS of the Vortex system. The list has been shortened with irrelevant algorithms for this work and extended with newer research in that field.

In our case we are looking for a proof which imposes ideally the same amount of work to an attacker and a valid sender. Since an attacker need a lot more resources than a normal sender the amount of work should be for him unbearable. A normal sender on the other hand should be able to cover the load imposed by the system easily.

The website coinwarz and [111] list the current mining speeds on up to date hardware for several algorithms as shown in table 5.2.

5.11.1. CPU-Bound Proof of work

CPU bound proof of work tries to assign a task to a request. By presenting a result the sender proofs that he solved a puzzle. Normally these puzzles may only be solved by brute force.

5.11.1.1. Hash based puzzle by Juels and Brainard

The hash puzzle by Juels and Brainard in [60] works as follows:

- The requester requests an operation.
- The service provider sends a block of data back with a known number of missing bits at the end and a cryptographic hash of the original block (including the missing bits)
- The requester then brute forces the missing bits until he finds the correct bit sequence to complete the hash puzzle and sends it to the service provider.

This puzzle has several downsides. One of the biggest downsides is that the puzzle may be solved by using multiple instances in parallel. This allows a specialised adversary to build an

efficient infrastructure for solving this kind of puzzle.

5.11.1.2. Hash based puzzle by Aura, Nikander and Leiwo

The hash based puzzle by Aura, Nikander, and Leiwo in [10] is an improved variant of [60]. It works as follows:

- The requester requests an operation.
- The service provider sends a nonce block N_{sp} to the requester and a number k (this number represents the difficulty).
- The requester chooses a nonce N_r .
- The requester brute forces $puzzlesolution$ by solving $hash(identity_r, N_{sp}, N_r, puzzlesolution) = hashValueStartingWithKBitsOfZeroes$.
- The requester sends N_r and X it to the service provider.

The downsides of this puzzles are that it may be calculated in parallel. This allows a specialised adversary to build an efficient infrastructure for solving this kind of puzzle. Furthermore the client side nonce N_r must be stored as long as the server side nonce N_{sp} is not regenerated in order to avoid replay attacks.

5.11.1.3. Modified Repeated Squaring Puzzle

Jeckmans proposed in [57] a Squaring hash puzzle. The algorithm is described as follows:

- $Setup(\ell)$: the sever generates a new public parameter sp with length ℓ bits using a (pseudo)random number generator. It then selects two random large primes p, q and creates $n = p \times q$ out of this. The server also selects a difficulty for the puzzles, k . The algorithm then outputs $mk = 2^k \bmod \phi(n)$ and $params = \langle k, sp, n \rangle$.
- $PuzzleGen(mk, req)$: $puz = \emptyset$ and $info = \emptyset$.
- $PuzzleSol(puz)$: $g = H(\langle sp, id_C \rangle)$, $sol = g^{2^k} \bmod n$.
- $PuzzleVer(info, mk, sol)$: This algorithm computes the generator, $g = H(\langle sp, id_C \rangle)$. The solution is then verified $sol \stackrel{?}{=} g^{2^k} \bmod \phi(n) \bmod n$.

| Criteria | Speed | hardness type | Cryptocurrency |
|-------------------------|------------|-----------------------|---|
| Proof of Work | | | |
| Blake-256[9] | 11 200 000 | FIXME | Blakecoin |
| CryptoNight | 2 | Memory or egalitarian | Monero |
| EtHash[37] | 108 000 | Memory and IO | Ethereum, Ethereum-classic |
| Equihash[17] | 1 | FIXME | Zclassic, Zcash |
| Groestl[45] | 45 000 | FIXME | Groestlcoin, Diamond |
| HEFTY1 | unknown | FIXME | Heavycoin, Mjollnircoin |
| Kecak[15] | 1 000 000 | FIXME | Maxcoin, Startcoin, CryptoMeth |
| Lyra2Rev2 [101] | 30 000 | Memory | Vertcoin |
| NeoScrypt | 400 | Memory | Feathercoin, UFOCoin, OrbitCoin, Phoenixcoin |
| SHA-256 | 9 460 000 | CPU | UniversalCurrency, Mazacoin, Titcoin, Peercoin, Bitcoin, HAMRadioCoin, eMarl, Digitalcoin-SHA-256, Zetacoin, Curecoin, Freicoi |
| Scrypt | 110 000 | Memory | Tigercoin, Terracoin, Unbreakable, Myriadcoin-SHA-256, Joulecoin, Acoin, Tekcoin, Unobtanium, Namecoin, DevCoin, IxCoin |
| Scrypt-Adaptive-NFactor | unknown | FIXME | Einsteinium, Gulden, Casinocoin, Quatloo, GameCredits, DNotes, Auracoin, Novacoin, CHNCoin, Goldcoin, Florincoin, Megacoin |
| Scrypt-Jane | unknown | FIXME | Bottlecaps, Litecoin, Bitbar, DigiByte, Verge, Franko, BATA, Worldcoin, WildBeastBitcoin, BBQCoin, Nyancoin, Digitalcoin-Scrypt |
| Quark | 20 000 | FIXME | Aricoin, Sexcoin, Spots, Emerald, Dodgecoin, Viacoin, Myriadcoin-Scrypt |
| X11 | 34 400 | FIXME | MonetaryUnit, Quark, Securecoin |
| X13 | unknown | FIXME | Digitalcoin-X11, Dash, CanabisCoin, darkCoin, HiroCoin, X11Coin |
| Yescrypt [80] | unknown | ROM-port | MaruCoin, BoostCoin, X13Coin |
| | | | Globalboost-Y |

Table 5.2.: comparison of Proof of Work algorithms

- $\text{BatchVer}(\text{info}, \text{mk}, \text{sol}_i | 1 \leq i \leq m)$: assume that the server has stored m solutions to check, $\text{sol}_i, 1 \leq i \leq m$. Compute the corresponding generators $g_i = H(\langle \text{sp}, \text{id}_{C_i} \rangle), 1 \leq i \leq m$. These solutions are then verified with batch verification using Equation $(\prod_{i=1}^m g_i)^{2^k} \bmod n \stackrel{?}{=} \prod_{i=1}^m \text{sol}_i \bmod n$.

This puzzle is a typical CPU bound puzzle compensating a lot of the drawbacks of [10] and [60] mentioned in the sections before. First it offers a potential verification in batches under the assumption that all solutions within the batch are correct. Secondly it is non-parallelizable. There is very little information to be stored on the server side and none of the information has a lifetime which is longer than the puzzle.

5.11.1.4. Hinted Hash Puzzle

In [40] the authors Feng, Kaiser, and Luu proposed a protocol and is closely related to [10]. The protocol itself is working as follows:

- The requester requests an operation and includes a nonce N_c .
- The service provider sends a chosen puzzle P and the hash answer $H(\langle A, N_s \rangle)$ to the requester.
- The requester brute forces A .
- The requester sends A to the service provider.

In the original protocols hashes and nonces are resent in order to verify if the requests do still match the originally sent data. As they have no

impacts on the mathematical side of the operation they have been eliminated in this description.

From today's view this protocol is however not very secure. Solutions may be searched in parallel and the protocol is susceptible to replay attacks as found result A is replayed over the course of time to re-authenticate

5.11.1.5. Chained Hash Puzzle

[50] (CPU Bound)



5.11.1.6. Targeted Hash Puzzle

[41] (CPU Bound)



5.11.1.7. Repeated Squaring Puzzle

[88] (CPU Bound)



5.11.1.8. Discrete Logarithm Puzzle

The puzzle of Waters et al. in [108] are somewhat similar to a brute force attack in a DH key exchange. This puzzle is non-parallelizable and is related to the hint based puzzles.



5.11.1.9. Subset sum puzzle

[103] (CPU bound)



5.11.3. Summary for Proof of Work Algorithms



[51]



[52]



[103]

5.11.2. Memory-Bound Proof of work

5.11.2.1. Moderately hard, memory-bound functions

[11] (Memory hard)



[4]



[33]

5.11.2.2. Memory bound puzzle based on pattern databases

[32] (Memory hard)



[104]



[27]



[2]



5.11.2.3. Argon2



5.12. Pseudo Random Number Generators



Scrypt was originally developed by Percival in 2009 and formally defined by Josefsson and Percival in [59]. It was originally designed as a key derivation function for an online backup service called tarsnap. According to the tarsnap site (<http://www.tarsnap.com/scrypt.html>) the cryptocurrency Litecoin uses a simplified version of the scrypt as proof of work scheme.



5.13. Known Attacks

In the following sections we emphasize on possible attacks to an anonymity preserving protocols. In the following sections we describe classes of attacks. 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.

5.11.2.5. HashCash



5.13.1. 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 kind of attacks. A good protocol must be able to react to such threads in a timely manor. This reaction should not rely on a required update of the infrastructure. The grade of security should solely be controlled by the users.

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

5.13.2. Attacks Targeting Anonymity

All attacks targeting users anonymity is a main focus of this work. Many informations may be leaked and the main goal should therefore rely on the principles established in security.

- Prevent an attack

This can only be done for attacks which 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 a valid protocol usage (this excepts bandwidth depletion attacks) be able to block malicious identities. Since establishing new identities is costly an attacker 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 this is not done by the implementation 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

discovering attacks (such as a query attack) may be done. However – we consider active attacks just as part of the ordinary message flow as an immediate result.

- Recover from an attack

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

In the following subsections we list a couple of attack classes which have been used against systems listed in 5.10 or the respective academic works. We list the counter measures which have been taken to deflect these attacks.

5.13.2.1. Probing Attacks

Identifying a node by probing and check their reaction is commonly done when fingerprinting a service. As a node is participating in a network and relaying messages probing may not be evaded. However, it may be made costly for an adversary to do systematic probing. This should be taken into account.

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). An early retry of the same identity would lead to automatic blacklisting. This is however an insufficient strategy as a huge adversary may have lots identities in stock. Requesting a particularly 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 a hash following certain 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 receiving address



5.13.2.2. Hotspot Attacks



5.13.2.3. Message Tagging and Tracing



5.13.2.4. Side Channel Attacks



5.13.2.5. Timing Attacks



5.13.2.6. Sizing Attacks



5.13.2.7. Bugging Attacks

Numerous attacks are available through 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 kind of proof of identities, such as certificates, offer some kind of 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. This is however not desirable from the security point of view.

Another possibility is to only 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 own infrastructure. Using Tor would violate the "Zero Trust" principle we are aiming for. Furthermore this measure would only conceal the source of the verification. It would not hide the fact that the message is being processed.

A fourth and most promising technology would be to force the sender of the certificate to include a "proof of non revocation". This could be a timestamped and signed partial CRL. Such a mandatory proof of non-revocation 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 a 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 mean time. It is recommended to keep the maximum cache time as low as 1d in order to avoid that revoked certificates may be used.

Bugging through DNS traffic: A widely common protocol in the internet is DNS. Almost all network related programs use it without thinking. Normally is the usage of such a protocol not a minor issue since the resolution of a lookup usually done by an ISP. In the case of a small 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 including 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 minimise the possibilities of this attack.

Bugging through external resources: An easy 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

5.13.3. Denial of Service Attacks

5.13.3.1. Censorship

Where as traditional censorship is widely regarded as selective information filtering and alteration a 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 is therefore not censorship resistant.

5.13.3.2. Credibility Attack

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

In a credibility attack a systems reputation is degraded in such a way that the system is no longer used. This may be achieved in several ways. This is usually done by reducing the reputation of a system.

This may be achieved in several ways. Examples:

- Disrupt functionality of a system.
This may be done by blocking ports the messaging protocol uses or blocking selectively messages. It may be done by removing publicly known participants from the internet either by law or by threatening.
- Publicly dispute the effectiveness of a system.
This is a very effective way to destroy a system. People are not willing to use a system which is compromised if the main goal of using the system is avoiding being observed.
- Reduce the effectiveness of a system.
A system may be considerably being loaded by an adversary to decrease the positive reception of the system. He may further use the system to send UBE to reduce the overall experience when using the system. Another way of reducing the effectiveness is to misuse the system for bad purposes and making them public.
- Dispute the credibility of the system founders.

Another way of reducing 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 an infrastructure is known or suspected to be run by a potential adversary peoples willingness to believe into such a system might be drastically reduced.

6. Applied Methodes

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

1. Identify problem hotspots for a new protocol.
2. Design a protocol which addresses the previously identified hotspots.
3. Build a protocol prototype.
4. Analyse the protocol for weaknesses using attack schemes.
 - a) Tagging/Bugging attacks
 - b) Tracing attacks

6.1. Problem Hotspots

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



6.1.1. Zero Trust Philosophy

One main disadvantage of almost any system listed in section 5.10 is that a trust (unlimited or limited) has been put into the infrastructure. In example when using Tor you need to trust the directory servers. Control over the directory servers might give an attacker the possibility to redirect a connection to controlled entry and exit nodes which would then break anonymity. In general control of entry and exit nodes makes a system vulnerable.

To avoid this problem we decided to apply a zero trust model. We do not trust any platform except for the sending and receiving computer. We assume that all other devices are compromised and do create detailed logs about what they are doing. We furthermore assume that traffic on the network layer is observed and recorded at any time. This philosophy creates very hard to meet goals.

But by assuming so we prevent the system from leaking information through side channels.

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

6.1.2. Information leakage

Information of messages or their meta data should not be leaked. This means that in a normal message flow any hop should be a valid sender, a valid recipient, or a valid mix this implies some kind of peer-to-peer (P2P) design.

6.1.2.1. P2P design

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

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

RQ2 (P2P): *Mixes and peers must be undistinguishable.*

6.1.2.2. Decoy traffic generation

In order to create decoy traffic in an untrusted way we need means to increase or decrease messages in size without knowledge of the routing node. An easy approach would be to create decoy traffic in the initial message. This would how-

ever create a pattern of decreasing message size in the net. To avoid this we introduced a set of operations to be applied to the original message. The operations are done in such a way that a mixer is unable to tell whether the message size or decrease results in decoy traffic generation/removal or not.

The main message operations are:

- Split a message in two parts.
- Merge two messages.
- Fork off a block xor'ed (create a new random block of the received message size, then use a bitwise xor of these blocks to create a new block. Send any or all of these blocks further).
- Join two blocks xor'ed (take two blocks and xor them together)

At this point we could have used homomorphic encryption instead of xor. This would however add a lot of complexity to the algorithm with no obvious gain. The bitwise xor however allows a cheap fast operation and it allows to split and join information at will. Furthermore the implications of this operation are well known and researched.

6.1.2.3. Message tagging or bugging protection

It is important to the protocol that any operation at any point of the protocol handling which is not foreseen should result in the failure of message transport. This makes the protocol very fragile but it prevents mixes from introducing tags which may be followed throughout the system.

In our approach we give a single mix the full control over the message hiding/blending layer. This means that every mix decides for the content there. However – This data is ephemeral and will (or may) be removed in the next node. The data received by a mix may be used to generate a “pseudo reply” on the blending layer to transport any other message (related or unrelated) back to the sending node. So tagging on this layer is worthless.

The reason for not giving control over the behaviour to this layer to the sender of the message is simple. By giving him control over it we would allow him to use the information provided here as the main medium. As an immediate re-

sult the system would suitable to blackmail any user of the world. It furthermore would create unintentional “exit nodes” to the system which might oppose further legal threads for participants.

RQ3 (untagable): *The message should be untagable (neither by a sender, or by an intermediate party such as a mixer).*

RQ4 (unbugable): *The message should be unbugable (neither by a sender, or by an intermediate party such as a mixer).*

6.1.2.4. Message replay protection

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

RQ5 (replay): *A message must not be replayable*

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

6.1.2.5. No Dedicated Infrastructure Philosophy

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

6.1.3. Accounting

The infrastructure must not be misused as UBE sending infrastructure. This implies that sending

messages is connected to some kind of “cost”. “Costs” must be connected to some kind of identity to allow accounting. Linking to a global identity would allow to assign traffic to a real world user. Therefore the protocol must allow to create ephemeral local identities not linked to a real identity.

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

6.1.4. Anonymisation

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

RQ7 (anonymisation): *System must be able to anonymise sender and recipient at any point of the transport layer and at any point of mixing unless it is the sender or the recipient itself.*

6.1.5. Initial Bootstrapping

The system must allow to boot strap from a zero knowledge or near zero knowledge point. Starting off from this point the protocol must be able to extend the network on its own.

RQ8 (bootstrapping): *The system must allow to boot strap from a zero knowledge or near zero knowledge point and extend the network on its own.*

6.1.6. Cypher selection

In This Protocol a lot of encryption and hashing algorithms have to be used. This Choice should be explained.

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

RQ9 (encryption variety): *The system must be able to use multiple symmetric, asymmetric, and hashing algorithms in order to immediately fall*

back to a secure algorithm for all new messages if required.

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

Secondly we need to have a spectrum of algorithm in such a manor that it may be (a) enlarged if necesary 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 favour 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 parameter.

The key sizes have been chosen in such a manor that the key types form tuples of approximately equal strength. The support of Camellia192 and Aes192 has been defined as optional. But as they are wildly common in implementations they have already been standardized as they build a possibility to step up security in future.

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

- Symetric
 - AES (key size: 128, 192, 256)
 - Camellia (key size: 128, 192, 256)
- Asymetric
 - RSA (key size: 2048, 4096, 8192)
 - Named Elliptic Curves
 - * secp384r1
 - * sect409k1
 - * secp521r1
- Hashing
 - sha384
 - sha512
 - tiger192

Within the implementation we assigned algo-

rithms to a security strength level:

- LOW
AES128, RSA1024
- MEDIUM
AES192, RSA2048, ECC secp384r1
- HIGH
RSA4096, ECC sect409k1, sha384
- QUANTUM
AES256, RSA8192, ECC secp521r1, sha512

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



6.1.7. Reed-Solomon function



6.1.8. Usability

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

RQ10 (easy handleable): *The system must be usable without cryptographic know-how and with well known tools.*

6.2. Protocol outline

The protocol itself is independently from the transport layer specified. We emphasize in this section to the general building blocks, the cryptographic structure, and the general protocol attributes. In section 9 we will then further elaborate the protocols inner structure.

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

1. Transport Layer

This Layer is provided by normal internet infrastructures. The main goal is to hide or blend our protocol into ordinary traffic within that layer.

2. Blending and subsequent layers

These layers may be provided by any user of the internet. Since these layers may be mixes only or valid endpoints they may or may not be publicly known. In a first implementation we build this system as a normal Java application. The main goal is to compile afterwards either native code and run it on a SoC like infrastructure such as a RaspberryPi or port it to an android device.

We may further split these layers into

a) Blending layer

This layer takes messages and creates transport layer conformant messages. In an ideal case the messages generated by this layer are undistinguishable from the normal message traffic and the embedded message is only visible for the receiving node.

b) Routing layer

The routing layer disassembles and re-assembles messages. This operation guarantees that messages are generated in such a way that decoy traffic is not differentiable from non decoy traffic.

c) Accounting layer

The accounting layer has three jobs. First he has to authorise the message processing after decrypting the header block. Secondly he handles all header request blocks and the reply blocks. And third, it keeps track of the accounting in regard to the sent messages.

6.2.1. Protocol Terminology

- **Sender:** The user or process originally composing the message.
- **Recipient:** The user or process destined to receive the message in the end.
- **Mix:** Any node processing the message. Please note all nodes are mixes.
- **Message:** The “real content” to be transferred from the sender to the recipient
- **Payload:** Any data transported between mixes regardless of the meaningfulness or relevance to the message.
- **Decoy traffic:** Any data transported between mixes which has no relevance to the

message at the final destination.

- **Identity:** A tuple of a routable address, and a public key.
- **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.



6.2.2. Vortex Communication model

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



6.2.3. Transport Layer

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

We identified in the previous section the following protocols as suitable for transport:

- SMTP
- XMPP

For a prototype we will implement a SMTP-Transport agent.

6.2.4. Blending Layer

The blending layer is taking care of multiple problems:

- It is translating the message block into a suitable format for transport
This includes jobs such as embedding a block as encoded text, as a binary attachment, or hide it within am message using steganography.
- Extract incomming blocks
Identify incomming messages containing a

possible block and extract it from the message.

- Do housekeeping on the storage layer of the transport protocol

It may be required that messages are deleted from time to time in order to stay below sizing quotas of an account.

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

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

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

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

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

6.2.5. Routing Layer

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

- Build structure representing the block building and the appropriate block IDs.
- Schedule all Routing blocks for processing in an priority queue.

- Authorise all routing blocks ready for processing with the calculated block sizes.
- Process blocks.
- Send prepared building blocks to the Blending layer.

after 5 days.



6.2.5.1. Block Structure

The main block structure is defined as a sequence of blocks.

The block sequence starts with a header containing a symmetric key encrypted with public key of the current node and a header block containing the immediate details to decrypt the subsequent blocks (if any).

The header block is followed by a routing block. This block contains information required for subsequent routing. According to this block currently valid data blocks may be assembled and sent to a different location.

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

The final sequence of blocks are data blocks. They contain the actual data or decoy traffic.

6.2.5.2. MURBs

The protocol allows to use MURBs. This enables a user to send a limited amount of times a message to an anonymous receiver without having any knowledge about its name or the location or infrastructure he is using.

A MURB in our term is a completely 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 has no knowledge about 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 it is used. In order to avoid statistical visibility we need to limit the number of uses per MURB. As a maximum number of usages the number 127 has been chosen. This number allows to be used for automated messages. A minute pattern would disappear latest after 2 hours and an hourly pattern

6.3. Protocol design

In this section we emphasize on the protocol blocks. These blocks are extracted from the blending layer and passed to the routing layer. A Vortex message is split into two main parts. The header block and the main block. The header block is divided into the two sub parts “headerKey” and “identity block”. Although those blocks are described in ?? as ASN.1 encoded structures they are not. In the message they are length prefixed fields.

In reality the structure is as follows:

- headerKey size as 16 bit unsigned int (big endian)
- headerKey bytes
- identityBlock size as 16 bit unsigned int (big endian)
- identityBlock bytes
- mainBlock size as 32 bit unsigned int (big endian)
- mainBlock bytes

The reason for not using ASN.1 encoding is that it might be possible to identify the unencrypted message on the transport layer as vortex message due to the ASN.1 structure. By switching to a length prefixed structure we make it impossible for an adversary to identify the unencrypted structures of a vortex message.



6.3.1. Header block

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

The header block contains the following data:

- A symmetric header key (header key)
This key is encrypted with the receiving identities public key. This key guarantees an efficient handling of the header data. As the number of bytes in the header is limited a

receiving node may assign all subsequent work to a known identity or discard it.

- An identity block (identityBlock)

This block contains a number of data which reflects the identity of the sender and the use of this (and subsequent blocks).

- sending ephemeral identity public key (identityKey)
- serial
- maximum number of replays for the serial for this identity
- Minimum number of seconds for replay protection.
- validity period for this block (in seconds)
- Symmetric decryption key for subsequent blocks
- Chain secret for the forwarding block
If specified the specified secret should be named in all subsequent blocks.
- Protocol requests to this node
- Identifier and padding for proof of work

- An identity signature (identitySignature)

Contains a signed hash (hash type is specified in identityBlock.hash) with the senders private key.

6.3.1.1. Ephemeral Identity

The identity in this header block is an ephemeral identity. It exists for a limited amount of time (recommended <90d). Creating a new ephemeral identity is done with an identity request. This identity is mapped to all accounting figures.

6.3.1.2. Requests

Requests are always embedded in a header block. All requests are answered with a provided MURB (which is recommended to have a maximum replay value of 1).

There are several header requests defined:

newIdentity request: This request may be answered with either a reject or a puzzle which is required to solve. Solving the puzzle results in

the creation of the identity on the node. Identities may be rejected by the node for various reasons:

- The node is not accepting newIdentity requests
- The identity is already taken.
- The identity is not strong enough (should be a longer key).
- The used encryption scheme is not supported by the node.

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

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

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

An identity is always ephemeral and expires after a given period of time. An identity may not be prolonged for security reasons. This oposes problems when using reply blocks. a reply block can only be valid as long as all included identities are valid. To counter this weakness without weakening security a ctxlessNewIdentity block may be sent to a reply block owner providing him with a new reply block.

queryPeer request: A peer request is a request for publicly known Vortex nodes. This request does offer the possiblility of harvesting the Vortex network. To counter this the following limitations should apply:

- The request should be very costly
- Only nodes advertising themselves as public are disclosed.
- Only one or two nodes should be disclosed upon request.
- The number of requests should be limited per identity.
- A node should always pick random nodes out of a 5% pool of known Vortex addresses.

These measures limit the effectivity of harvesting

attacks while giving a normal node the possibility of bootstrapping itself.

queryCapability request: This request is the only request answered as a clear text request. To minimize probing possibilities this request should be only answered if the node owner agrees or generally by public nodes.

messageQuota request: This request raises the number of routing blocks which may be processed for an identity. A node may reject this request depending on the load of the node, personal preferences, or because this identity causes too much traffic.

It is normally answered for all valid identities only. The node may answer it for recently expired identities. It is however not recommended to send a reply to an unknown identity as this behaviour might be used for probing of a node.

transferQuota request: This request raises the number of bytes which may be transferred for an identity. A node may reject this request depending on the load of the node, personal preferences, or because this identity causes too much traffic.

It is normally answered for all valid identities only. The node may answer it for recently expired identities. It is however not recommended to send a reply to an unknown identity as this behaviour might be used for probing of a node.

queryQuota request: This request instructs the node to send information about the given identity.

It is normally answered for all valid identities only. The node may answer it for recently expired identities. It is however not recommended to send a reply to an unknown identity as this behaviour might be used for probing of a node.

6.3.1.3. Replies to Clear Text Requests

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

This should only be a problem when bootstrapping or when adding new identities to the own address book.

6.3.2. Main block

The main two block types are routing blocks and payload blocks.

Routing blocks contain an onionised route chosen by the builder of the routing blocks and may contain instructions for building a message.

Payload blocks contain an encrypted message, parts of it or simply decoy traffic.

6.3.2.1. Routing Blocks

Routing blocks contain the following information:

- The node specification of the next hop (requires a full identity; may be not there if no next hop)
- Purpose of routing block (May be normal/statusOK/statusError)
- The moment of processing as a range in seconds since the time of arrival.
- Retention time in seconds since the time of arrival.
- The identity block for the next hop
- The routing block for the next hop
- Payload ID to be included
- Payload building instructions (optional; only if MURB)
- A signing key for the payload (optional; only if MURB)

6.3.2.2. Payload Building Instructions

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

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

Every node maintains a list of received blocks including their IDs and building instructions for them. Any node must keep blocks and building

instructions during the whole lifetime of a routing block. It may keep it longer. If a conflicting building instruction arrives all conflicting older rules are removed. Building instructions are always kept on a per-identity base. It is not possible to reference building instructions of a foreign identity.

splitPayload: Split a message block into two parts of variable size. The size is expressed in percent of the original block size.

mergePayload: Concatenates two messages of different or equal sizes.

xorMergePayload: Split a messsage block into two parts horizontally using an xor operation. Technically the operation works as follows:

- The size of the source block is determined.
- Create a new reproducible random pattern of data with the determined length.
- The original block is xor'ed with the random pattern block generated in the previous step
- Both blocks may be processed further and must be sent to different locations.

With this operation we may create decoy traffic, or split a message into two parts.

xorSplitPayload: This operation takes two payload chunk blocks and uses an xor operation to join them. Blocks with different sizes are processed.

encryptPayload: Encrypts a payload chunk block with a given symmetric key and algorithm. Please note that this operation changes the size of a message due to the keysize and the padding.

decryptPayload: Decrypts a payload chunk block with a given symmetric key and algorithm. Please note that this operation changes the size of a message due to the keysize and the padding.

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

are messages increasing the size as well as decreasing the size.

6.3.2.3. Reply Block

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

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

Replies are being built as an ordinary message blocks. To identify a Vortex message it must begin with the string “!vortex!” encoded in ASCII followed by a valid reply block structure. No additional bytes may be appended. Blocks with additional data should be discarded. To express a block starting with “!vortex!” the token is repeated twice.

replyCapability block: The reply contains the following information:

- Vortex identities (not encoded)
- Supported Vortex transports including blending specification.
- Maximum quotas.
- Minimum identity sizes.
- Supported cyphers and hashes.
- Maximum number of simultaneous valid header serials.
- Maximum number of simultaneous valid building operations.
- Maximum identity lifespan in seconds.



ReplyPuzzleRequired

block:

replyStatus block: General answer block signalling a status. The block is limited in length in order to minimize misuse of bandwidth. The Block contains the following data:

- Three digit status number
- Sending node identity

- Status text (optional)
- Affected block ID (optional)

- It protects from replaying .
- It offers an ephemeral identity with a limited lifespan.

ctxlessNewidentity block: This block may be used to signal the change of identity to a recipient. As this request is signed with the old known identity no means should exist to hijack such an identity.

This request contains:

- old Identity
- new Identity
- Signature (with old identity)

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

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

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

It is important that the accounting described here is for mixes. A node assembling messages needs to keep a lot more information.

6.3.2.4. payload Block

The payload block contains the actual message or decoy traffic. Since this block is heavily modified in course of the transport of the block it is built very simple.

It contains:

- A block ID
- The payload data
- A payload signature (optional; required for vortex blocks)

It is important to understand that a block may be exchanged at any time by an evil node. This however does not affect the safety on the message.

Any tagging introduced at this point does invalidate the stream. The output after the next hop is completely unpredictable.

6.3.3.1. Accounting Data of an Identity

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

The following data has to be kept per identity:

- **ID** $\langle pubKey, expiry, messagesLeft, bytesLeft \rangle$
- **Puzz[]** $\langle expiry, request, puzzle \rangle$
- **Build[]** $\langle expiry, targetID, buildOperation \rangle$

The **ID** tuple is the longest living tuple. It reflects an 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.

Puzz[] is a list of not yet solved puzzles of this user. Every puzzle has a relatively short lifespan (typically below 1d). As the server may decide how many outstanding puzzles he accepts he controls explicitly the size of this.

Build[] is a list of building instructions for a message. The server may decide at any time to reject a too big list or a too 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.

6.3.3. Accounting

Accounting covers several purposes in this system:

- It makes the system costly for nodes sending bulk messages.

6.3.3.2. Accounting Data of a “header” Block

All accounting data of a header block is connected to the respective identity. All header blocks do expire latest with their respective identity. The following data has to be kept on a mix:

- $\text{HL}\langle\text{identity}, \text{serial}, \text{remainingReplays}, \text{expiry}\rangle$

This is the long term header list. It lists all headers of an identity, and how many replays are left until the requests are rejected.

- $\text{HS}\langle\text{identity}, \text{serial}, \text{arrivalTime}, \text{hash}, \text{duration}\rangle$

This is the short term replay protection. It protects a block with the same hash from being replayed.

6.4.2. Transmission Block Processing



6.5. Sub Research Questions Roundup

6.5.1. SQ1: Sending messages maintaining unlinkability against an adversary



6.5.2. SQ2: Attacking unlinkability and circumvention



6.3.3.3. Accounting Data of a “payload” Block

All accounting data is connected to a header block and expires latest with its header block. The expiry time is however relative to the arrival time of the header block.

6.5.3. SQ3: Attack Mitigation by design



- $\text{PL}\langle\text{identity}, \text{serial}, \text{id}, \text{hash}, \text{expiry}, \text{content}\rangle$

The *content* may either be stored as the content or as the building instructions of the content as it is not always clear what block will be required for sending. A block may be defined for diagnostic paths only.

6.4. Protocol handling

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

6.4.1. Received Block Processing

A Block picked up in the blending layer is handled as shown in image 6.1.





Figure 6.1.: flow diagram showing processing of incomming messages

Part II.

Results

To verify the hypothesis made in this paper, and to analyse properties of the protocol in a real world scenario a library was implemented in Java which was capable of handling all message packets and the routing stack as a whole. The following paragraphs describe the protocol developed in general as a generic approach. Appendix ?? gives the full ASN.1 representation of the protocol.

It is important to notice that ASN.1 has no mean to express encrypted structures. Due to this fact we defined all encrypted fields as OCTET STRING.

The protocol is defined int the ASN.1 to support onionized information in an unencrypted form. This is meant for debugging purposes only. At no point should this possibility be used in a production environment.

The protocol described in the next chapter is independent from routing. We built a blending layer for SMTP. Layers for other protocols such as XMMP may be built similarly. The protocol may be extended by adding blending layers and their addressing schemes.

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

7. Protocol Overview

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

The data transferred is passed thru a number of mixes. The builder of a routing block (normally the sender) decides upon the following attributes:

- Hops for the message and all decoy traffic.
- Timing behaviour respectively speed of the message.
- Decoy traffic generation.
- Set of possible recipients.

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

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

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

A mix always applies the operations requested in the building instructions to the received data. If

this is not done properly the message may fail to transfer to its final destination.

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

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

7.1. Vortex Message

The outline in figure 7.1 is a simplified view onto the message block of MessageVortex.

A vortex message is a message passed from one mix to another this message may or may not contain any valuable information. The message itself is embedded as binary data into a transfer protocol. Every Mix may decide for himself what kind of protocols and embedding mechanisms are supported.

A vortex message is always built out of these blocks:

- header
 - Encrypted message key.
Contains symmetrical key for decryption of follow up header information and payload blocks. Encrypted with host identity public key.
 - Identity and “proof of work” information
Contains Requests, Proof of work sections, and header key
- Encrypted payload blocks (encrypted with header key)
 - Routing blocks (encrypted with message key)
 - * Next hop timing instructions
 - * Next hop routing blocks (encrypted)
 - * Next hop header

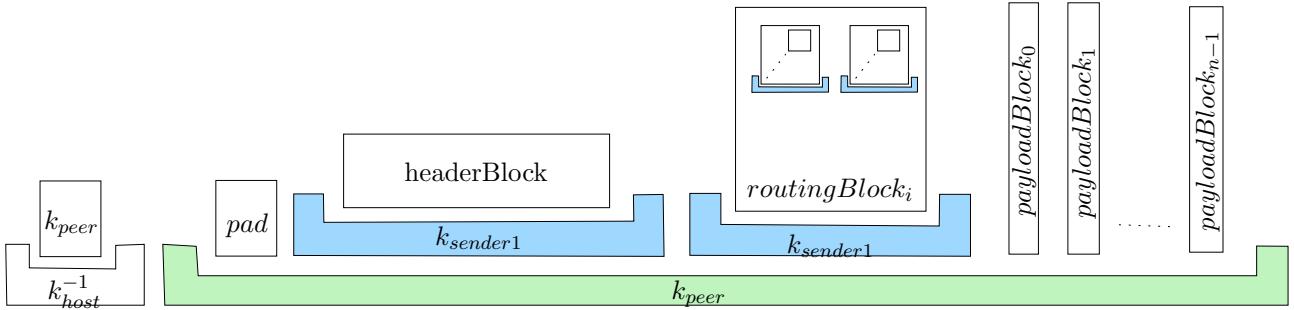


Figure 7.1.: Simplified protocol outline

- * Message build instructions.
- * Next hop header key and spec.
- * Next hop blending instructions.
- Payload chunk blocks

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

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

The second key is the header key located in the encrypted header. This key was chosen by the creator of the routing block containing the information. This key protects the inner structure of the Message. it makes it impossible for any node except the sending or the receiving node to detect the inner structure of the message. Without this key any independent observer with knowledge about the blending capabilities of a receiving node may;

- easier identify the block structure.
This remains the case regardless whether ASN.1 or length prefixed structures are used. If the structure of a vortex Message can be easily identified the messages may be logged or dropped.
- Identify the routing block size.
The value of this information is only very limited as it only reflects indirectly the complexity of the remaining routing information.
- Identify the number of payload blocks and

their respective sizes.

This is valuable information when following a traffic.

7.1.1. Key Usage

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

7.1.1.1. Peer key

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

7.1.1.2. Header Key

The header key is a symmetric key protecting the routing and ephemeral identity information of the message. It is prepended to the header section and protected by the identity key of the processing node.

7.1.1.3. Identity Key

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

Depending on the usecase (eg. unlikable signatures or scalable security) a user may use multiple keypairs at the same time.

7.1.1.4. Ephemeral Identity Key

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

7.1.2. VortexMessage Processing

In order to process messages some FP arithmetic is required. To make sure that all implementations on all platforms behave exactly the same we always use arithmetic as specified in ??.

7.1.2.1. Receiving Messages

All messages are processed as follows:

1. Extract length prefixed message key from block. Abort if not decryptable or invalid block.
2. Decrypt message key with hosts private key → message key
3. Extract length prefixed header message. Abort if too big.
4. Decrypt header block with decrypted message key. Abort if not decryptable or invalid block.
 - Verify identity
 - Check quotas (if any)
 - Extract header key
 - Extract requests (if any)
 - Check replays (if any)
5. Decide if message should be processed. If not abort here.
6. Decrypt main block with message key
7. Extract payload chunks.
8. Decrypt routing blocks with header key.
9. Check backref secrets.

Every routing block creates a new message.

The payload of a message is generated according instructions in the routing block. Timing instructions are relative to the arrival time of the message containing the routing block. This is

necessary as a routing block may be used multiple times (see section 6.2.5.2).

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

7.1.2.2. Building and Sending Messages

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

The message is then built as follows:

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

7.1.3. VortexMessage Operations

All operations are expressed as described in section 6.3.2.2. The following sections give important details about the implementation of the operations.

7.1.3.1. VortexMessage 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
Floatingpoint number describing the size of the first chunk. If fraction is 1 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:

$$\text{split}(f, pb_1) = \langle pb_1, pb_2 \rangle \quad (7.1)$$

$$pb_1.\text{startsWith}(pb_2) \quad (7.2)$$

$$pb_1.\text{endsWith}(pb_3) \quad (7.3)$$

$$\text{len}(pb_2) = \text{floor}(\text{len}(pb_1) \cdot f) \quad (7.4)$$

$$\text{len}(pb_1) = \text{len}(pb_2) + \text{len}(pb_3) \quad (7.5)$$

$$\text{xorSplit}(pb_1, f, pi) = \langle pb_2, pb_3 \rangle \quad (7.10)$$

$$pb_2 = \text{prng}(i, \text{floor}(\text{len}(pb_1) \cdot f)) \quad (7.11)$$

$$pb_3 = pb_1 \oplus pb_2 \quad (7.12)$$

$$\text{len}(pb_2) = \text{floor}(\text{len}(pb_1) \cdot f) \quad (7.13)$$

$$\text{len}(pb_3) = \max(\text{len}(pb_1), \text{len}(pb_2)) \quad (7.14)$$

For details about the implemented PRNGs see 5.12.

7.1.3.2. VortexMessage MergePayload Operation

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

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

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

$$\text{merge}(pb_1, pb_2) = pb_3 \quad (7.6)$$

$$pb_3.\text{startsWith}(pb_1) \quad (7.7)$$

$$pb_3.\text{endsWith}(pb_2) \quad (7.8)$$

$$\text{len}(pb_3) = \text{len}(pb_1) + \text{len}(pb_2) \quad (7.9)$$

7.1.3.3. VortexMessage XorSplitPayload Operation

The xorSplitPayload operation forks payload block into two payload blocks. The parameters for this operation are:

- Source payload block pb_1
- fraction f
- PRNG Initializer pi

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

7.1.3.4. VortexMessage XorMergePayload Operation

The xorSplitPayload operation forks payload block into two payload blocks. The parameters for this operation are:

- First source payload block pb_1

- Second source payload block pb_2

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

$$\text{xorMerge}(pb_1, pb_2) = pb_3 \quad (7.15)$$

$$pb_3 = pb_1 \oplus pb_2 \quad (7.16)$$

$$\text{len}(pb_3) = \max(\text{len}(pb_1), \text{len}(pb_2)) \quad (7.17)$$

7.1.3.5. VortexMessage EncryptPayload Operation

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

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

The operation follows the following rules (please note section 1.2.1 for notation):

$$\text{encrypt}(pb_1, spec, k) = pb_2 \quad (7.18)$$

$$pb_2 = E_{\text{spec}}^{K_a}(pb_1) \quad (7.19)$$

$$\text{len}(pb_2) \geq \text{len}(pb_1) \quad (7.20)$$

7.1.3.6. VortexMessage DecryptPayload Operation

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

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

The operation follows the following rules (please note section 1.2.1 for notation):

$$decrypt(pb_1, spec, k) = pb_2 \quad (7.21)$$

$$pb_2 = D_{spec}^{K_a}(pb_1) \quad (7.22)$$

$$\text{len}(pb_2) \leq \text{len}(pb_1) \quad (7.23)$$

binary representation of all blocks and data see chapter 9.

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

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

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

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

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

7.1.3.7. VortexMessage addRedundancy Operation



7.1.3.8. VortexMessage removeRedundancy Operation



7.2. VortexMessage Request Processing

Vortex message requests allow a Vortex node to gain knowledge about the Vortex network and create new identities.



7.2.1. VortexMessage Requests

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

The request information is defined in section 6.3.1.2. For more information about the exact

7.2.1.1. QueryCapability Request

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

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

7.2.1.2. NewIdentity Request

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

This request generates a temporary ephemeral identity for the limited time denoted in the validity field of the reply block. No quota is assigned during that phase. As soon as the identity offers

a valid puzzle solution the requested quota is assigned and the identity may be used for subsequent requests.

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

Request with too high message or transfer quota requests should be answered with an replyPuzzleRequired block containing a zero length puzzle.

7.2.1.3. QueryPeer Request

As outlined in section 6.3.1.2 this request should be very costly and harvesting of known addresses should be hard. Furthermore this request should always disclose the nodes from a fixed subset of the nodes known to the queried mix. This maximizes the effort to harvest participating nodes.

It is at the same time worth mentioning that this limit may pose a threat that traffic is concentrated on similar nodes within participating nodes using the same initial set of known addresses to bootstrap. This due to the fact that if using the same node to bootstrap for multiple participants within a group results in peer knowledge which is similar. thus resulting in a similar network. It is however also worth mentioning that participants belonging to more than one group will evolve over time as their different peer partners will result in different anonymity sets over time even when applying exact the same and reproducible rules.



7.2.1.4. TransferQuota, MessageQuota, and QueryQuota Request

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

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

The Message Quota is a quota not limiting the

number of bytes but the number of messages. As every message generates accounting overhead this number has to be limited as well. There are constraints similar to transferQuota when raising this value.

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

7.2.2. VortexMessage Reply Blocks

Reply blocks are as outlined in section 6.3.2.3 prefixed payload blocks. All reply blocks except the ctxless... reply blocks are a result of a header request.

7.2.2.1. ReplyCapability block

The replyCapabilityBlock is the reply block to a queryCapability Request. The information provided here is outlined in section 6.3.2.3. It is important to note that this block is even when requested in plain is always onionized and thus unreadable for third parties.

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

7.2.2.2. replyPuzzleRequired block

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

Every puzzle block will create an accounting entry as outlined in section 6.3.3.

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

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



7.2.2.3. replyStatus block



7.2.2.4. CtxlessNewidentity block

The new identity block is to propagate a newly generated, non-ephemeral identity of a user. This block should not be ignored by any node. It enables a user to strengthen its identity.

It is however should ignore the request if its acceptance results in a reduced security level of the current identity.

8. Vortex Prerequisites

8.1. Hardware

No special Hardware is required for running Vortex nodes. The capabilities of Vortex are designed 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 use a RaspberryPi Version 1. It is however recommended to use a faster, newer model due to the memory requirements of the proof of work.



8.1.1. Considerations Regarding Weak or Unsuitable Hardware



8.2. Addresses

A Vortex address is built as follows:
vortexAddress = < transport >:< address >! < publicKey >

To allow storage of Vortex addresses in standard messaging programs such as Outlook or Thunderbird. We define an alternate representation *encodedVortexAddress = base64(< transport >:< address >). < publicKey > @localhost.*

The suffix “@localhost” makes sure that a message intended for Vortex is not routed by any non-participating server.

The main downside of vortex addresses are 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 main email address for the public key. Such requests must then be answered by the vortex account with the valid Vortex address.

8.2.1. Public Key Encoding in Address Representation

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 representation is then encoded using BASE64

8.3. Transport Layers

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

8.3.1. SMTP

In SMTP we defined “smtp” (lower case) to be the transport specifier. When using SMTP messages are always encoded as attachments.

If not further specified by the receiving node an attachment should be the encoded message.

Valid properties may be:

- Encoding ”encoding = ”(”plain : ” < numberOfBytesOfOffset > [, < numberOfBytesOfOffset >] * |”f5”)

All SMTP nodes responding to clear text should at least support *encoding = plain : 0, 1024*.

8.3.2. XMMP



8.4. Client

We did not create a Vortex client for sending messages. Instead we used a standard Thunderbird email client pointing to a local SMTP and IMAP Server provided by a Vortex proxy. On the SMTP side Vortex does encapsulate where possible mails into a Vortex message and builds automated route to the recipient. The SMTP part of vortex may be used to encapsulate automatically all messages with a known Vortex identity into a vortex message. 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.

This has certain downsides. At the moment the vortex client has only a local store this makes it impossible to handle multiple simultaneously connected clients to use Vortex. This is however just a lack of the current implementation and not of the protocol itself as we may safely use an IMAP storage for storing vortex mails centrally.

8.4.1. Vortex Accounts

By definition any transport layer address may represent a Vortex identity. This fact may make people believe that their current email or jabber address is suitable as Vortex address. This 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.
- If a vortex node is malfunctioning non-vortex messages should remain unaffected. This is more likely to happen if non-Vortex messages are kept in a separate account.
- If a user wants no longer to maintain its Vortex address (hopefully there will be a better technology in future) he just may give up his Vortex accounts. If he would have been using his normal messaging account for Vortex he would receive mixing messages which he has to filter in future.

8.4.2. Vortex Node Types

8.4.2.1. Public Vortex Node

Public nodes are nodes which advertise themselves as mixes. Just as all nodes they may be an endpoint or a mix. Typically they accept all requests exactly as outlined in 9.1. As an immediate result of the publicly available information about such a node the owner may be target of our adversary. Pressure may be applied 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 (on the same or a different infrastructure).

8.4.2.2. Stealth Vortex Node

This node does not answer any clear text requests. As an immediate result the node is only usable by other nodes knowing the public key of this node. The node is therefore on a known secret base only reachable.

8.4.2.3. Hidden Vortex Node

A hidden node is a special form to a stealth node it has a set of preset identities. Only these entities are processed. This behaviour has certain drawbacks. An identity may not be changed. As an immediate result traffic may become a pseudonymity. To counter this effect at least partially we may use the same local identity for multiple senders. As an immediate result the sender is only one of all senders knowing the private key of an identity.

9. MessageVortex - Transport Independent Messaging anonymous to 3rd Parties

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

We use common messaging protocols as transport layers and connect to them using the respective client protocols. The actual mixes are operated by the users on their “always connected” devices. It goes without saying that such a system is far less reliable than a traditionally run server as this hardware is typically cheap and normally connected to the internet using a bandwidth shared media.

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

As decoy traffic is generated with the same operations as the true content is split it is impossible for an adversary running a node to determine whether he is generating noise or actually processing the true message.



9.1. Protocol Implementation



9.2. Block Structure

9.2.1. Header Block



9.2.2. Routing Blocks



9.2.3. Payload Blocks



9.3. Message Building



9.4. Accounting

In table 9.1 we show under what circumstances a reply to a header request should be sent. The capitalised words MAY, MUST, SHOULD and SHOULD NOT are used as defined in RFC2119[21].



9.5. Blending layer

In our implementation the blending layer is a weak implementation. It attaches a message

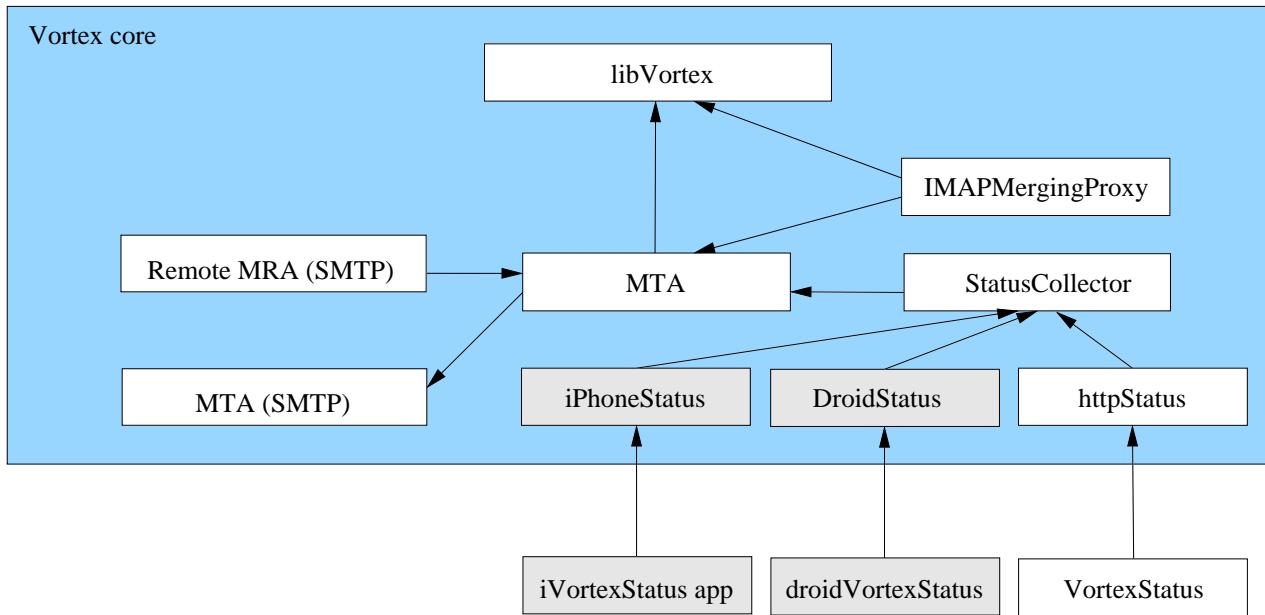


Figure 9.1.: Overview of the Vortex modules

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

Table 9.1.: Requests and the applicable criteria for replies

in non-obfuscated form with a mime type application/octet stream the application name is randomly chosen. Start offset is fixed to 200 and a valid header part is prepended. Text is not generated.



9.5.2. F5 Steganography



9.6. Message Flows



9.7. Considerations for Building Messages

9.5.1. Plain Inclusion



9.5.1.1. File Type Candidates



In a worst case scenario we assume that an adversary is controlling most of the network utilized for anonymisation. While this is not necessarily a problem (as pointed out earlier) it allows an adversary to track a message while agents are being used under his control. So for simplicity and as a worst case assumption we always assume that an adversary has perfect knowledge of an associated message flow. This is however a worst case scenario. One missing agent disconnects

the whole chain and as messages are not traceable in size.



9.7.1. Ephemeral identities



9.7.2. Timing of messages



9.7.3. Diagnostics



9.7.3.1. Implicit Diagnostic



9.7.3.2. Automatic Explicit Diagnostic



9.7.3.3. On-Demand Explicit Diagnostic



9.8. Verification of requirements

In the previous sections we identified the following list of requirements:

RQ1 No infrastructure should be trusted unless it is the senders or the recipients infrastructure.

RQ2 Mixes and peers must be undistinguishable.

RQ3 The message should be untagable (neither by a sender, or by an intermediate party such as a mixer).

RQ4 The message should be unbugable (neither by a sender, or by an intermediate party such as a mixer).

RQ5 A message must not be replayable

RQ6 The system must be able to do accounting without being linked to a real identity.

RQ7 System must be able to anonymise sender and recipient at any point of the transport layer and at any point of mixing unless it is the sender or the recipient itself.

RQ8 The system must allow to boot strap from a zero knowledge or near zero knowledge point and extend the network on its own.

RQ9 The system must be able to use multiple symmetric, asymmetric, and hashing algorithms in order to immediately fall back to a secure algorithm for all new messages if required.

RQ10 The system must be usable without cryptographic know-how and with well known tools.

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



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

This requirement is therefore fulfilled.

RQ2 (P2P): No node has additional privileges or offers additional services. All are equal and share the same privileges.

This requirement is therefore fulfilled.

RQ3 (untagable): Messages may not be tagged. All content is either strictly onionised or defined and linked with unknown hooks. Tampering with a message will cause the message delivery to fail at the next node.

This requirement is therefore fulfilled.

RQ4 (unbugable): There are always means to bug a message. As we put trust in sender and recipient and we know already that a intermediate mixing node is not able to modify the message the protocol is hard to bug. There may be a possibility to bug a message with a routing log entry over DNS. If a recipient is not resolving names or trusts in the content of such a message he is safe.

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

RQ5 (replay): Messages may only be replayed a limited amount of times. The number of replays is controlled by the sender and may not be altered by any mix. A malfunctioning mix replaying more often than allowed will not be able to extract any information than the information it obtained when sending the first time.

This requirement is therefore fulfilled.

RQ6 (accounting): All identities generated are not traceable as any identity is generated without any context and may not be mapped to an older or newer identity (perfect anonymous forward identity). Neither the source nor the replies may be used to be traced as all messages

This requirement is therefore fulfilled.

RQ7 (anonymisation): This point is the hardest to proof. We certainly achieved a high degree of anonymity. No node can tell by observing traffic if a node is a final recipient or just a mix. There are however some weaknesses in the protocol. As the implementation is currently connecting simultaneously to the true name (email or jabber account in the current implementation) and the Vortex account the user might be identified by that fact. Using a anonymisation proxy could solve the problem but it would violate the Zero trust principle.

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



(bootstrapping):



(encryption

variety):



(easy

handleable):

9.9. Considerations for Routing Messages

9.9.1. Time of sending

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

9.10. Real World Considerations

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

9.10.1. No Routing Log

There was up until very late a routing log functionality in the protocol. This functionality did however have the disadvantage that it allowed bugging and could possibly disclose intermediate mixes to a recipient which did not comply with the policy the mixes might have chosen. Therefore this

feature was dropped and replaced with the fetch block behaviour.

9.10.2. No Verification Possibilities

9.10.3. Very Limited Control Blending Layer for the sending node

9.10.4. Message Content



10. Security Analysis

In the following sections we emphasize on attacks targeting either sender recipient tuples or on identification of participants.

Based on the protocol we may safely assume the following key points:

- An adversary knows and controls a significant number of nodes (for our analysis we assume less than 80%).
- An adversary may observe the traffic at any point without getting any information about the message content
- An adversary is not capable of matching multiple messages on different nodes to one message.

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

- We assume that an adversary knows all messages of a transaction running over his nodes and matches them correctly to the same message.
-

We assume that the adversary is targeting the following informations:

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

10.1. Attack on Users Identity



10.1.1. Frequency and Bandwidth Analysis



10.2. Attack on Message content



10.2.1. Attacking Routing Blocks



10.3. Attack on Message size



11. Additional Considerations

11.1. Man in the Middle Attacks to Conversations



11.2. Identification of Participating Accounts



11.2.1. Identification by Content

It is possible to identify a message by content. Assuming that an adversary knows the applied blending methode he may identify an ASN.1 structure of



11.2.2. Identification by Query



11.2.3. Identification by Traffic Type



11.2.4. Identification by Bandwidth



11.2.5. Identification by Behavioural Analysis



11.3. Storage of Messages and queues

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

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



11.4. Economy of transfer



Part III.

Discussion

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

We first apply a statical analysis of the protocol to identify all informations leaked at all levels.

Then we apply a dynamic analysis of the protocol to identify all meta informations leaked during transmission of the protocol.

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

12. Static analysis

In this section we analyze statically the protocol. Looking at a full message we get the Protocol outline as shown in (12.1) on page 82.

12.3. Intermediate node routing layer



12.1. Transport and Blending Layer

12.4. Receivers routing layer

12.1.1. Identifying a Vortex Message Endpoint

Depending on the blending method, single messages might be identified as long as they are detectable. Detectability depends on various factors such as:

- Broken internal file structure (due to plain blending)
- Uncommon high entropy in a structureless file
- Unrelated message flow (see [55])
- Non-human behaviour on the transport layer (e.g., message traffic 24x7)

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

Assuming a global observer as an adversary and unencrypted traffic, he might discover the originating routing layer and thus identifying it as a Vortex node by following traces of the transport layer.

12.2. Senders routing layer



$$VortexMessage = \langle \mathbf{P}^{K_{hostN}^{-1}}, \langle \mathbf{PAD}, \mathbf{CP}^{K_{hostN}^{-1}}, \mathbf{H}^{K_{senderN}}, [\mathbf{R}^{K_{senderN}}], [\mathbf{PL}] * \rangle^{K_{peerN}} \rangle \quad (12.1)$$

$$\mathbf{P}^{K_{hostN}^{-1}} = E^{K_{hostN}^{-1}}(\mathbf{PREFIX}\langle K_{peerN} \rangle) \quad (12.2)$$

PAD = {32 padding bytes from payload} (12.3)

$$\mathbf{CP}^{K_{hostN}^{-1}} = E^{K_{hostN}^{-1}}(\mathbf{CPREFIX}\langle K_{senderN} \rangle) \quad (12.4)$$

$$\mathbf{H}^{K_{senderN}} = E^{K_{senderN}}(\mathbf{HEADER}) \quad (12.5)$$

HEADER = (12.6)

$$\mathbf{R}^{K_{senderN}} = E^{K_{senderN}}(\mathbf{ROUTING}) \quad (12.7)$$

ROUTING = (12.8)

PL = (12.9)

Figure 12.1.: Detailed representation of a VortexMessage

13. Dynamic analysis

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

13.1. Attacks against the vortex system itself

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

13.1.1. DoS Attacks against the System

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

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

13.1.1.1. Traffic Replay

Traffic replay is a common way to highlight traffic in many systems by replaying the same traffic and increase through this the signal to noise ratio of the system.

Due to the replay protection of the vortex protocol it is not possible to of subsequent messages as the messages are terminated on the first conformal behaving node.

13.1.2. Diagnosability of traffic

13.1.2.1. Hijacking of Header and Routing Blocks

In order to Hijack a header or routing block, an attacker needs to know the forward secret which is contained within the encrypted data. Probability to generate such a

13.1.2.2. Partial Implicit Routing Diagnosis



13.1.2.3. Partial Explicit Routing Diagnosis



13.1.2.4. End-to-End Routing Diagnosis



13.1.2.5. Denial of Service by Exhausting Quotas or Limits



13.2. Achieved Anonymity and Flaws

13.2.1. Measuring Anonymity



13.2.2. Attacking Anonymity through Traffic Analysis

As traffic and decoy traffic and decoy traffic are chosen by the creator of the routing block frequency patterns can not be detected, unlike the router did create them. Same applies to message sizes and traffic hotspots.

13.2.7. Attacking Anonymity through Replay Analysis



13.2.8. Attacking Anonymity through Tamper Replay Analysis



13.2.2.1. Attacking Anonymity through Timing Analysis



13.2.2.2. Attacking Anonymity through Throughput Analysis



13.2.3. Attacking Anonymity through Routing Block Analysis



13.2.4. Attacking Anonymity through Header Analysis



13.2.5. Attacking Anonymity through Payload Analysis



13.2.6. Attacking Anonymity through Bugging



14. Recommendations on using the Vortex Protocol



15. Achieved Goals and Weaknesses

Although the protocol was carefully designed it has certain flaws. These flaws typically assume that parts of the underlying security has been severely broken to be exploited.

15.1. Message content

Although it is possible to embed any type of content into a Vortex message great care should be taken as content may allow to disclose a readers identity or location. For this reason only self contained messages should be used (such as plain text messages).

15.1.1. Splitting of message content



15.1.2. Redundancy



15.1.3. Redundancy Detection as Attack Pattern



15.1.4. routing Considerations



15.1.5. Hotspot Avoidance



16. Anonymity



16.0.1. Size of the Anonymity Set



16.0.2. Jurisdictional implications onto the Anonymity Set



16.1. Effects of anonymous communication on behaviour



Part IV.

Appendix

The RFC draft document

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: March 26, 2019

Gwerder
September 22, 2018

MessageVortex Protocol
draft-ietf-messagevortexmain-00

Abstract

MessageVortex Protocol specifies messages embedded within existing transfer protocols such as SMTP or XMPP to send them anonymously from peer to peer.

The protocol outperforms other protocols by decoupling transport from the final transmitter and receiver party. There is no trust put into any infrastructure except for the infrastructure of the sending and receiving party of a message. The Message flow is entirely selected by the creator of the routing block. Routing nodes gain no non-obvious knowledge about messages even when collaborating. Third-party anonymity is always achieved. Furthermore, one out of sender and receiver anonymity may be achieved.

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 March 26, 2019.

Copyright Notice

Copyright (c) 2018 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

Gwerder

Expires March 26, 2019

[Page 1]

(<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 | 4 |
| 1.1. Requirements Language | 5 |
| 1.2. Protocol Specification | 5 |
| 1.3. Number Specification | 5 |
| 2. Entities Overview | 5 |
| 2.1. Node | 5 |
| 2.1.1. NodeSpec | 5 |
| 2.1.1.1. NodeSpec for SMTP nodes | 6 |
| 2.1.1.2. NodeSpec for XMPP nodes | 6 |
| 2.2. Peer Partner | 6 |
| 2.3. Encryption keys | 6 |
| 2.3.1. Identity Keys | 7 |
| 2.3.2. Peer Key | 7 |
| 2.3.3. Sender Key | 7 |
| 2.4. Vortex Message | 7 |
| 2.5. Message | 8 |
| 2.6. Key and MAC specifications and usages | 8 |
| 2.6.1. Asymmetric Keys | 9 |
| 2.6.2. Symmetric Keys | 9 |
| 2.7. Blocks | 9 |
| 2.8. Transport Address | 9 |
| 2.9. Identity | 10 |
| 2.9.1. Peer Identity | 10 |
| 2.9.2. Ephemeral Identity | 10 |
| 2.9.3. Official Identity | 10 |
| 2.10. Workspace | 10 |
| 3. Layer Overview | 11 |
| 3.1. Transport Layer | 11 |
| 3.2. Blending Layer | 11 |
| 3.3. Routing Layer | 12 |
| 3.4. Accounting Layer | 12 |
| 4. Vortex Message | 12 |
| 4.1. Overview | 12 |
| 4.2. Message Prefix Block (MPREFIX) | 13 |
| 4.3. Inner Message Block | 13 |
| 4.3.1. Control Prefix Block | 13 |
| 4.3.2. Control Blocks | 14 |
| 4.3.2.1. Header Block | 14 |

| | |
|--|----|
| 4.3.2.2. Routing Block | 14 |
| 4.3.3. Payload Block | 15 |
| 5. General notes | 15 |
| 5.1. Supported Symmetric Ciphers | 15 |
| 5.2. Supported Asymmetric Ciphers | 15 |
| 5.3. Supported MACs | 16 |
| 5.4. Supported Paddings | 16 |
| 5.5. Supported Modes | 16 |
| 6. Blending | 17 |
| 6.1. Blending in Attachments | 17 |
| 6.1.1. PLAIN embedding into attachments | 17 |
| 6.1.2. F5 embedding into attachments | 17 |
| 6.2. Blending into an SMTP layer | 18 |
| 6.3. Blending into an XMPP layer | 18 |
| 7. Routing | 18 |
| 7.1. Vortex Message Processing | 18 |
| 7.1.1. Processing of incoming Vortex Messages | 18 |
| 7.1.2. Processing of Routing Blocks in Workspace | 19 |
| 7.1.3. Processing of Outgoing MessageVortex Messages | 21 |
| 7.2. Header Requests | 21 |
| 7.2.1. Request New Ephemeral Identity | 21 |
| 7.2.2. Request Message Quota | 21 |
| 7.2.3. Request Increase of Message Quota | 21 |
| 7.2.4. Request Transfer Quota | 22 |
| 7.2.5. Query Quota | 22 |
| 7.2.6. Request Capabilities | 22 |
| 7.2.7. Request Nodes | 22 |
| 7.2.8. Request Identity Replace | 23 |
| 7.3. Special Blocks | 23 |
| 7.3.1. Error Block | 23 |
| 7.3.2. Requirement Block | 23 |
| 7.3.2.1. Puzzle Requirement | 24 |
| 7.3.2.2. Payment Requirement | 24 |
| 7.4. Routing Operations | 24 |
| 7.4.1. Mapping Operation | 25 |
| 7.4.2. Split and Merge Operations | 25 |
| 7.4.3. Encrypt and Decrypt Operations | 25 |
| 7.4.4. Add and Remove Redundancy Operations | 25 |
| 7.4.4.1. Padding Operation | 26 |
| 7.4.4.2. Apply Matrix | 26 |
| 7.4.4.3. Encrypt Target Block | 27 |
| 7.5. Processing of Vortex Messages | 27 |
| 8. Accounting | 27 |
| 8.1. Accounting Operations | 27 |
| 8.1.1. Time-Based Garbage Collection | 27 |
| 8.1.2. Time-Based Routing Initiation | 28 |
| 8.1.3. Routing Based Quota Updates | 28 |
| 8.1.4. Routing Based Authorization | 28 |

| | |
|--|----|
| 8.1.5. Ephemeral Identity Creation | 28 |
| 9. Acknowledgments | 28 |
| 10. IANA Considerations | 28 |
| 11. Security Considerations | 29 |
| 12. References | 31 |
| 12.1. Normative References | 31 |
| 12.2. Informative References | 33 |
| Appendix A. The ASN.1 schema for Vortex messages | 34 |
| A.1. The main VortexMessageBlocks | 34 |
| A.2. The VortexMessage Operation Structures | 37 |
| A.3. The VortexMessage Ciphers Structures | 39 |
| A.4. The VortexMessage Request Structures | 42 |
| A.5. The VortexMessage Replies Structures | 43 |
| A.6. The VortexMessage Requirements Structures | 45 |
| A.7. The VortexMessage Helpers Structures | 45 |
| A.8. The VortexMessage Additional Structures | 46 |
| Author's Address | 48 |

1. Introduction

Anonymization is hard to achieve. Most of the attempts in the past rely on either trust in a dedicated infrastructure or a specialized networking protocol.

Instead of defining a transport layer, MessageVortex piggybacks on other transport protocols. Messages are being transported alongside ordinary messages of that transport protocol. A blending layer picks the messages up, applies local operations to it and resends the new chunks to the next recipients.

A processing node learns as little as possible from the message due to the nature of the operations processed. The onionized structure of the protocol makes it impossible to follow the trace of a message without having control over the processing node itself.

MessageVortex is a protocol which allows sending and receiving messages by using a routing block instead of a destination address. 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 of the message transit. Decoy traffic and message traffic are not differentiable as the nature of the addRedundancy operation allows each generated part to be a message part or decoy. Any routing node is thus unable to differentiate between the message and decoy traffic.

Any Receiver knows after processing whether a message is destined for it (it creates a chunk with ID 1) or other nodes (processing instructions only). Due to the missing keys, no other node may do this processing.

This RFC starts with the general terminology (see Section 2). Next, a general overview of the process is given (see Section 3). Lastly, the subsequent sections describe the details of the protocol.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Protocol Specification

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

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 hex digits are octets in hex writing. A blank in ASCII (' ') is written as 20h and a capital 'K' in ASCII as 4Bh.

2. Entities Overview

Within this document, we refer to the entities as defined below.

2.1. Node

We use the term 'node' to describe any system connected to other nodes, and supporting the MessageVortex Protocol. A 'node address' is typically an email address, an XMPP address, or any other transport protocol identity supporting the MessageVortex protocol. Any address SHOULD include a public part of an 'identity key' allowing to transmit messages safely. One or more addresses MAY belong to the same node.

2.1.1. NodeSpec

Every addressable node is addressed in a unified format stored in a nodeSpec block. This RFC specifies transport layers for XMPP and SMTP. Adding transport layers requires to write an extension to this RFC.

2.1.1.1. NodeSpec for SMTP nodes

Typically a node specification is specified with the ASN.1 block NodeSpec (see Appendix A.7). To allow addressing of a vortex node with an ordinary client an alternative representation SHOULD be supported as defined below as smtpAlternateSpec (specification noted in ABNF as specified in [RFC5234]).

```
localPart      = <local part of address>
domain        = <domain part of address>
email          = localPart "@" domain
keySpec        = <BASE64 encoded AsymmetricKey; DER representation>;
smtpAlternateSpec = localPart ":" keySpec "@" domain
```

2.1.1.2. NodeSpec for XMPP nodes

Typically a node specification is specified with the ASN.1 block NodeSpec. To allow addressing of a vortex node with an ordinary client, an alternative representation SHOULD be supported as defined below as jidAlternateSpec (specification noted in ABNF as specified in [RFC5234]).

```
localPart      = <local part of address>
domain        = <domain part of address>
jid            = localPart "@" domain [ "/" resourcepart ]
keySpec        = <BASE64 encoded AsymmetricKey; DER representation>;
jidAlternateSpec = localPart ":" keySpec "@" domain [ "/" resourcepart ]
```

Please note that the ":" character (3Ah) is explicitly excluded in [RFC7622] in the localpart.

2.2. Peer Partner

We use the term 'peer partner' as two or more message sending or receiving entities. One of these peer partners sends a message, and all other peer partners receive one or more messages. Peer partners are message specific. Every peer partner always connects directly to a node.

2.3. Encryption keys

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

2.3.1. Identity Keys

Every participant of the network has an asymmetric key. These keys SHOULD be either EC keys with a minimum length of 384 bits or RSA keys with a minimum length of 2048 bits.

The public key needs to be known by all parties writing to or through that node.

2.3.2. Peer Key

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

A peer key is included in the identity block of a Vortex message and in the building instructions for a Vortex message (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 decouples further identity and processing information from the previous key.

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

2.4. Vortex Message

We use the term 'Vortex message' for a single transmission between two routing layers. A 'blended Vortex message' describes a Vortex message which has been adapted to the transport layer by the blending layer (see Section 3).

A full vortex message contains the following items:

- o The peer key (encrypted with the host key of the node; stored in a PrefixBlock) protecting the inner Vortex message (innerMessageBlock).
- o The small padding guarantees that a replayed routing block with different content does not look alike.

- o The sender key (encrypted with the host key of the node) protecting the identity and routing block.
- o The identity block (protected by the sender key) containing information about the ephemeral identity of the sender, replay protection information, header requests (optional) and a requirement reply (optional).
- o The routing block (protected by the sender key) containing information on how subsequent messages are processed, assembled and blended.
- o The payload block (protected by the peer key) contains payload chunks for processing.

2.5. Message

A Message is a content to be transmitted from one sender to the recipient. The Sender uses a routing block to achieve this which is either built by him or provided by the receiver. A Message may be anonymous. There are however different degrees of anonymity:

- o If the sender of a message is not known to anyone else except the sender, we refer to that as 'sender anonymity.'
- o If the receiver of a message is not known to anyone else except the receiver, we refer to that as 'receiver anonymity.'
- o If an attacker is unable to determine content, original sender, and final receiver, we refer to that as third-party anonymity.
- o If a sender or a receiver may be determined as "one of a set of $<k>$ entities we refer to it as k-anonymity (for more about this see [KAnon]).

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

2.6. Key and MAC specifications and usages

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

The following key structures exist:

- o SymmetricKey
- o AsymmetricKey

MAC does not need a complete structure containing specs and value. Instead only a MacAlgorithmSpec is available. The following sections outline the constraints when specifying parameters for these structures. A node MUST NOT specify any parameter more than once.

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

2.6.1. Asymmetric Keys

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

RSA and EC keys must contain a keysize parameter. All asymmetric keys SHOULD contain a padding parameter. A node SHOULD assume PKCS#1 if no padding is specified.

NTRU specification MUST provide parameters "n", "p", and "q". McEliece specification MUST contain parameters "n", "k", and "t".

2.6.2. Symmetric Keys

Nodes use symmetric keys for encrypting payload and control blocks. All symmetric keys MUST contain a key type specifying a strictly normed key. They MUST contain a key encoded.

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

2.7. Blocks

We use the term 'block' for an ASN.1 sequence in a transmitted message. We embed messages in the transport protocol. It may have any size.

2.8. Transport Address

We use the term 'transport address' for 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 transport address.

2.9. Identity

2.9.1. Peer Identity

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

- o A transport address (always) and the public key of this identity (given there is no recipient anonymity)
- o The routing blocks for contacting the identity (optional; this block is only required in the case of recipient anonymity)
- o The private key (only known by the owner of the identity)

2.9.2. Ephemeral Identity

Ephemeral identities are temporary identities created on a single node. These identities MUST NOT relate to any other identity. They allow bookkeeping for a node. To each ephemeral identity is a workspace assigned. Every ephemeral identity may have multiple quotas assigned.

2.9.3. Official Identity

An official identity may have the following items assigned:

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

2.10. Workspace

Every official or ephemeral identity has a workspace. A workspace consists of the following elements:

- o Zero or more routing blocks to be processed
- o Slots for payload block sequentially numbered. Every slot...
 - * MUST contain a numerical ID identifying the slot
 - * MAY contain a payload content

- * If a block contains a payload, it MUST contain a validity period.

3. Layer Overview

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

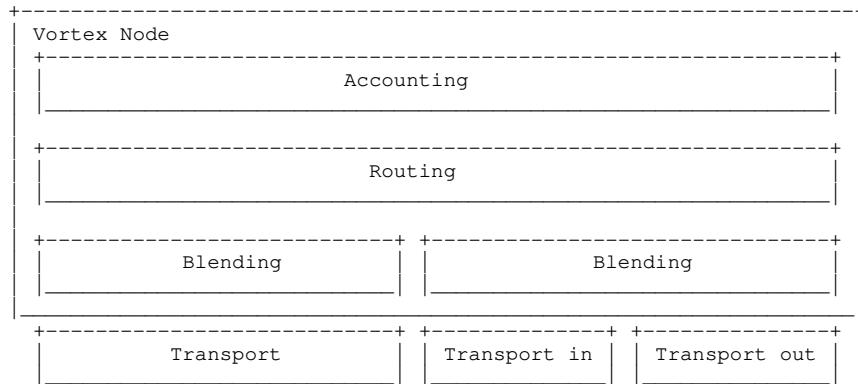


Figure 1: Layer overview

Every participating node MUST implement the layers 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 respective Transport layers for sending and receiving packets.

3.1. Transport Layer

The transport layer embeds the blended MessageVortex packets into the data stream of the existing transport layer protocol.

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

3.2. Blending Layer

The blending layer embeds MessageVortex packets into the transport layer data stream and extracts MessageVortex packets from the transport layer.

3.3. Routing Layer

The Routing Layer expands information contained in MessageVortex packets, processes them, and passes generated packets to the respective Blending Layer.

3.4. Accounting Layer

The accounting layer keeps track of all ephemeral identities authorized to use a MessageVortex node. Especially it 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. The three to four letter abbreviations denote the key required for decryption. The abbreviation k_h stands for the asymmetric host key. sk_p is the symmetric peer key. The receiving node obtains this key by decrypting MPREFIX with its host key k_h . sk_s is the symmetric sender key. When decrypting the MPREFIX block, the node obtains this key. The sender key protects the header and the routing blocks. This key guarantees that the node assembling the message does not know about upcoming identities, operations, and requests. The peer key protects the message including structure from any third party observer.

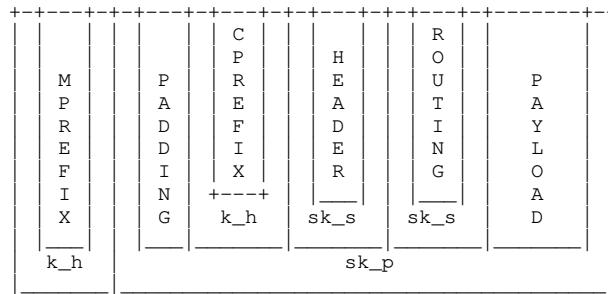


Figure 2: Vortex message overview

4.2. Message Prefix Block (MPREFIX)

The PrefixBlock contains a symmetrical key as defined in Appendix A.1 and is encrypted using the host key of the receiving peer host. The symmetric key used MUST be one out of 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. A prefix block MUST contain the same forward-secret as the other prefix, the routing block, and the header block. A host MAY reply to a message with an unencrypted message block. Any reply to a message SHOULD be encrypted.

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

4.3. Inner Message Block

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

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

An InnerMessageBlock contains so-called forwardSecrets. This random number MUST be the same in the HeaderBlock, the RoutingBlock, and the PrefixBlock. Nodes receiving Messages containing non-matching forwardSecrets MUST discard these messages, and SHOULD NOT send an error message.

4.3.1. Control Prefix Block

Control prefix block (CPREFIX) and MPREFIX block share the same structure and logic. It contains the sender key sk_s. If an MPREFIX block was unencrypted, a node MAY omit the CPREFIX block. An omitted

CPREFIX block results in unencrypted control blocks (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 contain the core information to process the payload. It contains a HeaderBlock and a RoutingBlock.

4.3.2.1. Header Block

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

- o It MUST contain the local ephemeral identity of the routing block builder.
- o It MAY contain header requests.
- o It MAY contain the solution to a PuzzleRequired block previously opposed in a header request.

The list of header requests MAY one of the following:

- o be empty
- o contain a single identity create request (HeaderRequestIdentity)
- o contain a single increase quota request

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

4.3.2.2. Routing Block

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

- o It MUST contain a serial number uniquely identifying the routing block of this user. A serial number MUST be unique during the lifetime of a routing block.
- o It MUST contain the same forward secret as the two prefix blocks and the header block.

- o It MAY contain assembly and processing instructions for subsequent messages.
- o It MAY contain a reply block for messages assigned to the owner of the identity.

4.3.3. Payload Block

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

5. General notes

The MessageVortex protocol is a modular protocol. It allows using different encryption algorithms. For operation, a Vortex node SHOULD always support at least two completely different (i.e. relying on two different mathematical problems) types of algorithms, paddings, or modes.

5.1. Supported Symmetric Ciphers

A node MUST support the following symmetric ciphers:

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

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

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

5.2. Supported Asymmetric Ciphers

A node MUST support the following asymmetric ciphers:

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

5.3. Supported MACs

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

- o SHA256 (see [ISO-10118-3] for SHA implementation details)
- o RipeMD160 (see [ISO-10118-3] for RIPEMD implementation details)

A node SHOULD support the following MACs:

- o SHA512
- o RipeMD256
- o RipeMD512

5.4. Supported Paddings

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

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

5.5. Supported Modes

A node MUST support the following modes:

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

A node SHOULD NOT use the following modes:

- o NONE (Except as stated when using the addRedundancy function)
- o ECB

A node SHOULD support the following modes:

- o CTR ([RFC3686])
- o CCM ([RFC3610])

- o OCB ([RFC7253])
- o OFB ([MODES])

6. Blending

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

The following sections describe the blending mechanism. There are currently two blending layers specified. One layer specification for the simple mail transfer protocol (SMTP; See [RFC5321]) and one for the Extensible Messaging and Presence Protocol (XMPP; See [RFC6120]).

6.1. Blending in Attachments

There are two types of blending supported when using attachments.

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

A node MUST support PLAIN blending for reasons of interoperability.
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. A routing node SHOULD use a credible payload type (e.g., MIME type) with high entropy. It furthermore SHOULD prefix a valid header structure to avoid easy detection of the Vortex message. A routing node SHOULD use a valid footer, if any, to a payload file to improve blending.

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

6.1.2. F5 embedding into attachments

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

6.2. Blending into an SMTP layer

Email messages containing messages MUST be encoded with Multipurpose Internet Mail Extensions (MIME) as specified in [RFC2045]. All nodes MUST support BASE64 encoding. A node 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 as specified 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 any credentials.

6.3. Blending into an XMPP layer

For interoperability, an implementation SHOULD provide XMPP blending.

Blending into XMPP traffic is done using the [XEP-0234] extension of the XMPP protocol.

PLAIN and F5 blending is 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 unauthenticated at first. 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:

- o A node MUST ignore all Routing blocks.
- o A node MUST ignore all Payload blocks.
- o A node SHOULD accept identity creation requests in unauthenticated messages.
- o A node MUST ignore any other header request except identity creation requests.

- o A node MUST ignore all identity creation requests which belong to an already existing identity.

A message is considered authenticated as soon as the identity used in the header block is known and not temporary. For authenticated messages the following rules apply:

- o A node MUST ignore identity creation requests.
- o 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 no reply block is provided.
- o A node SHOULD process all header requests.
- o A node SHOULD add all routing blocks to the workspace.
- o 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 and contains at least one payload block.

7.1.2. Processing of Routing Blocks in Workspace

A routing workspace consists of the following items:

- o The identity linked to it (This determines the lifetime of the workspace).
- o The routing combos (RoutingCombo) linked to it.
- o A payload chunk space. Multiple subspaces are available within this space:
 - * 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 represents 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 this identity.

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

- o All processed routing combos.
- o All routing combos with expired usagePeriod.
- o All payload chunks when exceeded their maxProcess time.
- o All expired objects.
- o All expired puzzles.
- o All expired identities.
- o 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. The trigger occurs at least minProcessTime after the last octet of the message arrived at the routing layer. A node SHOULD choose the latest possible moment in such a way that the peer node receives the last octet of the assembled message before maxProcessTime is reached. The calculation of the 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 identity of a single transport message.

First, the processing node escapes the payload chunk at ID 0 if needed (non-special block starting with a backslash). Next, it executes all processing instructions of a routing combo in the sequence specified. If an instruction fails, the block at the target ID of the operation remains unchanged. The routing layer proceeds with the subsequent processing instructions, 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, a VortexMessage is composed and passed to the blending layer.

7.1.3. Processing of Outgoing MessageVortex Messages

The blending layer MUST then 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 done 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 rejecting the request.

If a node accepts an identity creation request, it MUST send a reply. To accept a request without a requirement, an accepting node MUST send back a special block containing "no error". To accept a block with a requirement, an accepting node MUST send a special block containing a requirement block.

7.2.2. Request Message Quota

Any valid ephemeral identity may request to raise the current message quota to a specific value at any time. The request MUST include a reply block in the header. The request may contain other parts. If a requested value is lower than the current quota, 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. The increase is requested 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. 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, 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 raise the current transfer quota to a specific value at any time. The request MUST include a reply block in the header. The request may contain other parts. If a requested value is lower than the current quota, 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. The request may contain other parts.

A node MUST reply to a HeaderRequestQueryQuota request (see Appendix A). The reply 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. The capabilities include all information necessary to create a parseable VortexMessage. Any node SHOULD reply to any encrypted HeaderRequestCapability.

7.2.7. Request Nodes

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

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

7.2.8. Request Identity Replace

This request allows a receiving node to replace an identity with the identity provided in the message. This request is required if an adversary managed 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 validly authenticated message to another identity providing the new transport and key details.

A node SHOULD reply to a such a request of a valid known identity. The reply MUST include an error message or a "No Error" status message.

7.3. Special Blocks

Special blocks are payload messages. They 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 a backslash (5Ch) 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 where specified as a payload. 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 a human-readable text providing additional messages about the error.

7.3.2. Requirement Block

If a node is receiving a requirements block, it MUST assume that the request block has been accepted, has not been processed yet, and will be processed if the contained requirement is met. A node MUST process a request as soon as the requirement is fulfilled. A node MUST resend the request as soon as the requirement is met.

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

To solve a puzzle posed by a node a Vortex Message needs to be sent to the requesting node. This Vortex Message MUST contain a header block which includes the puzzle block and MUST have a MAC fingerprint starting with the bit sequence as specified in the challenge. A node calculates the MAC from the unencrypted DER encoded HeaderBlock with the algorithm specified by the node. To meet this requirement, a node adds a proofOfWork field to the HeaderBlock.

7.3.2.2. Payment Requirement

If a node requests a payment, 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 in which the payment may be carried out. A node MUST accept the payment if carried out within the 'valid' period but confirmed later. A node SHOULD return allunsolicited payments to the sending address.

7.4. Routing Operations

Routing operations are contained in a routing block and processed either on arrival on a message or when a compiling new message. All Operations are reversible. No Operation is available for generating decoy traffic. For decoy traffic, either encryption of an unpadded block may be used, or the addRedundancy operation.

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

When applying an operation to a source block, the resulting target block inherits the expiry of the of the source block. When having multiple different expiry times, the expiry the furthest in the future will be applied to the target block. If the operation fails, the target expiry remains unchanged.

7.4.1. Mapping Operation

A mapping operation is a straightforward operation mainly used in inOperations of a routing block to map the routing block 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 these constraints:

- o 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.
- o All calculations must be done according to IEEE 754 [IEEE754] and in 64 Bit precision.
- o If a relative value is a non-integer result, a floor operation (cutting off all non-integer parts) determines the number of bytes.
- o If an absolute value is negative, the size reflected applies to the number of bytes counted from the end of the message chunk.
- o If an absolute value is bigger than the number of bytes in a block, 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 below zero. An operation MUST fail if a relative value is equal to or above 100. All floating point operations must be carried out 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 before. An encryption operation encrypts a block symmetrically and places the result in the target block. The parameters MUST contain required parameters such as IV, padding, or 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 the core operations of the protocol. They may be used to split messages and

distribute message content across multiple routing nodes. The operation is split 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 in an addRedundancy operation. For a removeRedundancy operation invert the functions and order.

7.4.4.1. Padding Operation

First, the length of all output blocks including redundancy is calculated. This is done by $L=\text{roof}((\text{input block size in bytes})+4)/(\text{encryption block size in bytes}) * (\text{block size in bytes})$. The block is prepended with a 32-bit length indicator in bytes at the beginning (little-endian). This length indicator i is calculated by $i = \text{randominteger}() * L$. The rest of the input block up to length L is padded with random data. If GF(16) is applied, numbers are treated as little-endian representations.

For padding removal, first, the padding i at the start is removed as a little-endian integer. Then, 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 is as big as the subsequent encryption operation and does not require any further padding.

7.4.4.2. Apply Matrix

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

Next, we multiply the data matrix D by a Vandermonde matrix V . The V matrix has the number of rows equal to the incols calculated, and columns are equal to the $\text{number of data blocks}$. The content of the matrix is formed by $v(i, j) = \text{pow}(i, j)$, whereas i reflects the row number starting at 0, and j reflects the column number starting at 0. Please note that calculations noted here have to be carried out in

the GF noted in the respective operation to be successful. The operation results in a matrix A.

7.4.4.3. Encrypt Target Block

Each row vector of A is a new data block which is then encrypted with the corresponding encryption key noted in keys of the addRedundancyOperation. If there are not enough keys available, the keys used for encryption are reused from the beginning after the last key has been 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 information contained in a routing block in the workspace. All operations MUST be executed in the sequence provided in the routing block. Any failing operation must leave the result block unmodified.

All workspace blocks resulting in IDs 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 major kinds of operations:

- o Time-based operations (cleanup jobs and initiation of routing)
- o Routing triggered operations (updating quotas, authorizing operations, and pickup of incoming messages)

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

8.1.1. Time-Based Garbage Collection

The accounting layer SHOULD keep a list of expiry 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 any time 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 of time, 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. It MUST decrease message quota before processing routing blocks in the workspace. A node MUST decrease the message quota after the processing of any 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 all header requests have been processed. The message quota MUST be decreased by one on each routing block triggering processing in the workspace.

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. The request includes a lifetime of the ephemeral identity. The costs for creating the ephemeral identity SHOULD raise if a longer lifetime is requested.

9. Acknowledgments

Thanks go to my family which did support me with patience and countless hours and 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 may be understood more as a toolset than 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 did rely on the following focus areas:

- o Confidentiality
- o Integrity
- o Availability
- o Anonymity
 - * 3rd party anonymity
 - * sender anonymity
 - * receiver anonymity

All these factors are affected by the usage of the protocol. The following sections provide a list of factors affecting the primary goals.

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

If a transport layer supports encryption, 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 the routing block builder, but the presence of a vortex node message may be detected. This may be done either by detecting 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 easy detection by minimizing these effects.

A blending layer SHOULD use carrier files with high compression or encryption. Carrier files SHOULD NOT have inner structures so that the payload is comparable to valid content. To achieve undetectability by a human reviewer, a routing block builder should

use F5 blending instead of PLAIN blending. This, however, increases the protocol overhead roughly by a tenfold.

The two layers 'routing' and 'accounting' have the deepest insight into a Vortex message's inner working. They know the immediate peer sender and the peer recipients of all payload chunks. As decoy traffic is generated by combining chunks and applying redundancy calculations upon them, a node can never know whether a malfunction (e.g., when doing a recovery calculation) was intended or not. Therefore a node is unable to tell a failed transaction apart from a terminated transaction. It furthermore cannot tell content apart from decoy traffic.

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

- o All operations applied SHOULD be credibly involved in a message transfer.
- o There should always be a sufficient subset of the result of an addRedundancy operation sent to peers to allow recovery of the data built.
- o The anonymity set of a message should be sufficiently large to avoid legal prosecution of all jurisdictional entities involved. It has to be large enough to do so even if a certain amount of the anonymity set cooperates with an adversary.
- o Encryption and decryption SHOULD follow whenever possible normal usage. Avoid encrypting a block on a node with one key and decrypting it with a different key on the same or an adjacent node.
- o Traffic peaks SHOULD be uniformly distributed within the whole anonymity set.
- o A routing block SHOULD be used for a limited number of messages. If used as a message block for the node itself it should be used only once. A block builder SHOULD use the HeaderRequestReplaceIdentity block to update reply routing blocks on a regular base. Implementers should always keep in mind that the same routing block is identifiable as such by its structure.

An active adversary cannot use blocks from other routing block builders for his purposes. He may falsify the result by injecting wrong message chunks or by not sending a message. Such message disruptions may be detected by intentionally routing some information to the routing block builders' node. If the Vortex message does not

carry the information expected 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. A block building node MAY build redundancy paths into a routing block to withstand such malicious nodes.

A peer node may try to exhaust a routers quota by sending the same block multiple times. A block builder should take care that a block may not be resent in the interval where the previous blocks for the transaction are still valid. This guarantees that replayed blocks will stop at the next peer node. This provides better protection against replay based attacks.

Receiver anonymity is in danger if the handling of 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. Great care has to be taken when handling any 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, 11 1998.

[CCITT.X680.2002]

International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", 11 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", 10 2001.

[FIPS-AES]

Federal Information Processing Standard (FIPS), "Specification for the ADVANCED ENCRYPTION STANDARD (AES)", 11 2011.

[IEEE754]

IEEE, "754-2008 - IEEE Standard for Floating-Point Arithmetic", 08 2008.

- [ISO-10118-3] International Organization for Standardization, "ISO/IEC 10118-3:2004 -- Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions", 3 2004.
- [MODES] National Institute for Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", 12 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", 05 2009.
- [TWOFISH] Schneier, B., "The Twofish Encryption Algorithm: A 128-Bit Block Cipher, 1st Edition", 03 1999.
- [XEP-0234] Peter, S. and L. Stout, "XEP-0234: Jingle File Transfer", 08 2017, <<https://xmpp.org/extensions/xep-0234.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. 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>>.
- [RFC7622] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", RFC 7622, DOI 10.17487/RFC7622, September 2015, <<https://www.rfc-editor.org/info/rfc7622>>.

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
}

HeaderBlock ::= 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] HeaderBlock,
        -- 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 prefix to be used
    routing CHOICE {
        -- debug/internal use only
        plain [331] SEQUENCE (SIZE (0..maxRoutingBlocks))
            OF RoutingCombo,
        encrypted [332] SEQUENCE (SIZE (0..maxRoutingBlocks))
            OF OCTET STRING
    },
    -- 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
    messageKey      [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 Operation Structures

```

MessageVortex-Operations DEFINITIONS EXPLICIT TAGS ::=
BEGIN
    EXPORTS PayloadOperation;
    IMPORTS maxID
        FROM MessageVortex-Schema

```

```

SymmetricKey, AsymmetricKey, MacAlgorithmSpec, CipherSpec
    FROM MessageVortex-Ciphers
UsagePeriod, BlendingSpec
    FROM MessageVortex-Helpers;

-- maximum omega of the Galois field used
maxGFSIZE           INTEGER ::= 16
-- maximum size of a message chunk
maxChunkSize        INTEGER ::= 4294967295
-- minimum size of a message chunk (should be -maxChunkSize)
minChunkSize        INTEGER ::= -4294967295

PayloadOperation ::= CHOICE {
    splitPayload   [150] SplitPayloadOperation,
    mergePayload   [160] MergePayloadOperation,
    encryptPayload [300] EncryptPayloadOperation,
    decryptPayload [310] DecryptPayloadOperation,
    addRedundancy  [400] AddRedundancyOperation,
    removeRedundancy [410] RemoveRedundancyOperation
}

PercentSizeBlock ::= SEQUENCE {
    fromPercent      INTEGER (0..100),
    toPercent        INTEGER (0..100)
}

AbsoluteSizeBlock ::= SEQUENCE {
    fromAbsolute     INTEGER (0..maxChunkSize),
    toAbsolute       INTEGER (minChunkSize..maxChunkSize)
    -- negative toAbsolute denotes end in absolute value
    -- specified from the end of block
}

SizeBlock ::= SEQUENCE{
    size CHOICE {
        percent      [15001] PercentSizeBlock,
        absolute     [15101] AbsoluteSizeBlock
    }
}

AddRedundancyOperation ::= SEQUENCE {
    inputId          [16000] INTEGER (0..maxID),
    dataStripes      [16001] INTEGER (1..254),
    redundancy       [16002] INTEGER (1..254),
    keys             [16003] SEQUENCE (SIZE (2..512))
        OF SymmetricKey,
    outputId         [16004] INTEGER (1..maxID),
    gfSize           [16005] INTEGER (2..maxGFSIZE)
}

```

```

}

RemoveRedundancyOperation ::= SEQUENCE {
    inputId          [16000] INTEGER (0..maxID),
    dataStripes     [16001] INTEGER (1..254),
    redundancy      [16002] INTEGER (1..254),
    keys            [16003] SEQUENCE (SIZE (2..512))
                            OF SymmetricKey,
    outputId        [16004] INTEGER (1..maxID),
    gfSize          [16005] INTEGER (2..maxGFSIZE)
}

SplitPayloadOperation ::= SEQUENCE {
    originalId       INTEGER (0..maxID),
    firstSize        SizeBlock,
    newFirstId       INTEGER (1..maxID),
    newSecondId      INTEGER (1..maxID)
}

MergePayloadOperation ::= SEQUENCE {
    originalFirstId  INTEGER (0..maxID),
    originalSecondId INTEGER (0..maxID),
    newId            INTEGER (1..maxID)
}

EncryptPayloadOperation ::= SEQUENCE {
    originalId       INTEGER (0..maxID),
    key              SymmetricKey,
    newId            INTEGER (1..maxID)
}

DecryptPayloadOperation ::= SEQUENCE {
    originalId       INTEGER (0..maxID),
    key              SymmetricKey,
    newId            INTEGER (1..maxID)
}

END

```

A.3. The VortexMessage Ciphers Structures

```

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

    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 {
    xsadd         (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 {
    sha256        (3000),
    sha384        (3001),
    sha512        (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.4. The VortexMessage Request Structures

```

MessageVortex-Requests DEFINITIONS EXPLICIT TAGS ::=
BEGIN
    IMPORTS StatusBlock, ReplyNodes, ReplyCurrentQuota, ReplyCapability
            FROM MessageVortex-Replies
    RequirementBlock
            FROM MessageVortex-Requirements
    UsagePeriod
            FROM MessageVortex-Helpers;

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

```

```

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

HeaderRequestIdentity ::= SEQUENCE {
    period UsagePeriod
}

HeaderRequestQuota ::= SEQUENCE {
}

HeaderRequestNodes ::= SEQUENCE {
    numberOfNodes INTEGER
}

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

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

HeaderRequestCapability ::= SEQUENCE {
    period UsagePeriod
}

END

```

A.5. The VortexMessage Replies Structures

```

MessageVortex-Replies DEFINITIONS EXPLICIT TAGS :=
BEGIN
    EXPORTS StatusBlock, ReplyCurrentQuota, ReplyCapability,
            ReplyNodes;
    IMPORTS BlendingSpec, NodeSpec
            FROM MessageVortex-Helpers
    CipherSpec
            FROM MessageVortex-Ciphers;

StatusBlock ::= SEQUENCE {
    code          StatusCode
}

```

```
}
```

```
StatusCode ::= ENUMERATED {
```

```
    -- System messages
```

```
    ok                      (2000),
```

```
    quotaStatus              (2101),
```

```
    puzzleRequired           (2201),
```

```
    -- protocol usage failures
```

```
    transferQuotaExceeded   (3001),
```

```
    messageQuotaExceeded    (3002),
```

```
    requestedQuotaOutOfBand (3003),
```

```
    identityUnknown          (3101),
```

```
    messageChunkMissing      (3201),
```

```
    messageLifeExpired       (3202),
```

```
    puzzleUnknown            (3301),
```

```
    -- capability errors
```

```
    macAlgorithmUnknown      (3801),
```

```
    symmetricAlgorithmUnknown (3802),
```

```
    asymmetricAlgorithmUnknown (3803),
```

```
    prngAlgorithmUnknown     (3804),
```

```
    missingParameters        (3820),
```

```
    badParameters             (3821),
```

```
    -- Mayor host specific errors
```

```
    hostError                (5001)
```

```
}
```

```
ReplyNodes ::= SEQUENCE {
```

```
    node  SEQUENCE (SIZE (1..5))
```

```
    OF NodeSpec
```

```
}
```

```
ReplyCapability ::= SEQUENCE {
```

```
    cipher      SEQUENCE (SIZE (2..256)) OF CipherSpec,
```

```
    maxTransferQuota  INTEGER (0..4294967295),
```

```
    maxMessageQuota   INTEGER (0..4294967295),
```

```
    supportedBlendingIn SEQUENCE OF BlendingSpec
```

```
}
```

```
ReplyCurrentQuota ::= SEQUENCE {
```

```
    messages INTEGER (0..4294967295),
```

```
    size      INTEGER (0..4294967295)
```

```
}
```

```
END
```

A.6. 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.7. 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 ::= SEQUENCE {
        notBefore      [0]   GeneralizedTime OPTIONAL,
        notAfter       [1]   GeneralizedTime 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
    }

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

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

END

```

A.8. 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

Martin Gwerder
Untere Parkstrasse 9
Hausen, AG 5212
Switzerland

Phone: +41 56 202 76 81
Email: rfc@messagevortex.net
```

Glossary

adverser FIXME

Agent FIXME

EWS FIXME

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 [29]. 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) FIXME

LMTTP FIXME

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 (eg. supports folders). A Local Mail Store may be an authoritative store for mails or a “Cache Only” copy. It is typically not a queue.

mail server admin FIXME

MDA An MDA provides an uniform access to a Local Mail Store.

Remote MDA A Remote MDA is typically supporting a specific access protocol to access the data stored within a Local Mail Store .

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

MRA A Mail receiving Agent. This agent receives mails from a agent. Depending on the used protocol two subtypes of MRAs are available.

Client MRA A client MRA picks up mails in the server mail storage from a remote MDA. Client MRAs usually connect thru a standard protocol which 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 forwardes received Messages to a MTA for delivery and routing. A typical protocol supported by an Server MRA is SMTP

MS-OXMAPIHTTP FIXME

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

MTA A Mail Transfer Agent. This transfer agent routes mails between other components. Typically an MTA receives mails from an MRA and forwardes them to a MDA or MSA. The main task of a MTA is to provide reliable queues and solid track of all mails 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 mails from a local storage and allows a user to read existing mails, create and modify mails.

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.

[97, FIXME]

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).

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 mails in a server based store. Modifying Mails or even handling a complex folder structure is not doable with POP

Service FIXME

SMTP SMTP is the most commonly used protocol for sending mails across the internet. In its current version it has been specified in [61].

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

user FIXME

UBE FIXME

Bibliography

- [1] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. "Moderately hard, memory-bound functions". In: *ACM Transactions on Internet Technology (TOIT)* 5.2 (2005), pp. 299–327. URL: <http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/2.pdf> (cit. on p. 36). 
- [2] Mehmud Abliz and Taieb Znati. "A guided tour puzzle for denial of service prevention". In: *Computer Security Applications Conference, 2009. ACSAC'09. Annual.* IEEE. 2009, pp. 279–288. DOI: [10.1109/ACSAC.2009.33](https://doi.org/10.1109/ACSAC.2009.33) (cit. on p. 36). 
- [3] Luis von Ahn, Andrew Bortz, and Nicholas J. Hopper. "k-Anonymous Message Transmission". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*. Ed. by Vijay Atluri and Peng Liu. ACM Press, Oct. 2003, pp. 122–130. DOI: [10.1145/948109.948128](https://doi.org/10.1145/948109.948128). URL: <http://www.abortz.com/papers/k-anon.pdf> (cit. on p. 19). 
- [4] Joel Alwen and Jeremiah Blocki. *Efficiently computing data-independent memory-hard functions*. Tech. rep. Cryptology ePrint Archive, Report 2016/115, 2016 (cit. on p. 36). 
- [5] IS Amiri and MRK Soltanian. *Theoretical and Experimental Methods for Defending Against DDoS Attacks*. Syngress, 2015 (cit. on p. 34). 
- [6] AMQP v1.0. AMQP.org, Oct. 2011. URL: <http://www.amqp.org/confluence/display/AMQP/AMQP+Specification> (cit. on p. 16). 
- [7] Banks Andrew and Gupta Rahul. *MQTT*. en. OASIS, Apr. 2014. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/ mqtt-v3.1.1.pdf> (cit. on p. 16). 
- [8] Elias Athanasopoulos, Mema Roussopoulos, Kostas G. Anagnostakis, and Evangelos P. Markatos. "GAS: Overloading a File Sharing Network as an Anonymizing System". In: *Proceedings of Second International Workshop on Security, (IWSEC 2007)*. Nara, Japan, 2007. URL: <http://dcs.ics.forth.gr/Activities/papers/gas%20iwsec07.pdf> (cit. on p. 30). 
- [9] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. "Sha-3 proposal blake". In: *Submission to NIST* (2008) (cit. on p. 35). 
- [10] Tuomas Aura, Pekka Nikander, and Jussi Sipekka Leiwo. "DOS-resistant authentication with client puzzles". In: *International workshop on security protocols*. Springer. 2000, pp. 170–177 (cit. on pp. 34, 35). 
- [11] Adam Back. "The hashcash proof-of-work function". In: *Draft-Hashcash-back-00, Internet-Draft Created,(Jun. 2003)* (2003). URL: <http://www.hashcash.org/docs/hashcash.html> (cit. on p. 36). 
- [12] Marco Valerio Barbera, Vasileios P. Kemerlis, Vasilis Pappas, and Angelos Keromytis. "CellFlood: Attacking Tor Onion Routers on the Cheap". In: *Proceedings of ESORICS 2013*. Sept. 2013. URL: <http://www.cs.columbia.edu/~vpk/papers/cellflood.esorics13.pdf> (cit. on p. 28). 
- [13] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. "Low-Resource Routing Attacks Against Tor". In: *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*. Washington, DC, USA, Oct. 2007. DOI: [10.1145/1314333.1314336](https://doi.org/10.1145/1314333.1314336). URL: <http://systems.cs.colorado.edu/~bauer/papers/wpes25-bauer.pdf> (cit. on p. 28). 
- [14] Mike Belshe, Roberto Peon, and Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 2015. DOI: [10.17487/rfc7540](https://doi.org/10.17487/rfc7540). URL: <https://rfc-editor.org/rfc/rfc7540.txt> (cit. on p. 15). 
- [15] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak sponge function family main document". In: *Submission to NIST (Round 2)* 3 (2009), p. 30 (cit. on p. 35). 
- [16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. "Argon2: new generation of memory-hard functions for password hashing and other applications". In: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2016, pp. 292–302. URL:

- <https://www.cryptolux.org/images/0/06/Argon2-euro.pdf> (cit. on p. 36). 
- [17] Alex Biryukov and Dmitry Khovratovich. "Equihash: Asymmetric proof-of-work based on the generalized birthday problem". In: NDSS. 2016. URL: <https://www.internetsociety.org/sites/default/files/blogs-media/equihash-asymmetric-proof-of-work-based-generalized-birthday-problem.pdf> (cit. on p. 35). 
- [18] Alex Biryukov, Ivan Pustogarov, and Ralf Philipp Weinmann. "TorScan: Tracing Long-lived Connections and Differential Scanning Attacks". In: *Proceedings of the European Symposium Research Computer Security - ESORICS'12*. Springer, Sept. 2012. URL: <http://freehaven.net/anonbib/papers/torscan-esorics2012.pdf> (cit. on p. 28). 
- [19] Alex Biryukov, Ivan Pustogarov, and Ralf Philipp Weinmann. "Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization". In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. May 2013. URL: <http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf> (cit. on p. 28). 
- [20] Carsten Bormann, Klaus Hartke, and Zach Shelby. *RFC7252: The Constrained Application Protocol (CoAP)*. RFC 7252. June 2014. DOI: 10.17487/rfc7252. URL: <https://rfc-editor.org/rfc/rfc7252.txt> (cit. on p. 17). 
- [21] S. Bradner. *RFC2119 Key words for use in RFCs to Indicate Requirement Levels*. IETF, 1997. URL: <http://tools.ietf.org/pdf/rfc2119.pdf> (cit. on p. 67). 
- [22] *Campaign Monitor*. 2012. URL: <http://www.campaignmonitor.com/resources/will-it-work/email-clients/> (cit. on p. 33). 
- [23] David Chaum. "Untraceable Electronic Mail, Return, Addresses, and Digital Pseudonyms". In: *Communications of the ACM* (1981). URL: http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/chaum81.pdf (cit. on pp. 20, 26, 28). 
- [24] David Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: *Journal of Cryptology* 1 (1988), pp. 65–75. URL: <http://www.cs.ucsb.edu/~ravenben/classes/595n-s07/papers/dcnet-jcrypt88.pdf> (cit. on p. 27). 
- [25] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System". In: *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. July 2000, pp. 46–66. URL: <https://freenetproject.org/> (cit. on p. 29). 
- [26] Fabien Coelho. *Exponential Memory-Bound Functions for Proof of Work Protocols Technical Report A/370/CRI version 3*. Cryptology ePrint Archive, Report 2005/356. <http://eprint.iacr.org/2005/356>. 2005 (cit. on p. 36). 
- [27] Fabien Coelho. "An (almost) constant-effort solution-verification proof-of-work protocol based on merkle trees". In: *International Conference on Cryptology in Africa*. Springer. 2008, pp. 80–93 (cit. on p. 36). 
- [28] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: Accountable Anonymous Group Messaging". In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: ACM, 2010, pp. 340–350. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866346. URL: <http://doi.acm.org/10.1145/1866307.1866346> (cit. on p. 30). 
- [29] M. Crispin. *RFC3501 INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. IETF, 2003. URL: <http://tools.ietf.org/pdf/rfc3501.pdf> (cit. on p. A51). 
- [30] Norman Danner, Sam DeFabbia-Kane, Danny Krizanc, and Marc Liberator. "Effectiveness and detection of denial of service attacks in Tor". In: *Transactions on Information and System Security* 15.3 (2012), 11:1–11:25. DOI: 10.1145/2382448.2382449. URL: <http://arxiv.org/pdf/1110.5395v3.pdf> (cit. on p. 28). 
- [31] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. URL: <https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt> (cit. on p. 28). 
- [32] Sujata Doshi, Fabian Monroe, and Aviel D Rubin. "Efficient memory bound puzzles using pattern databases". In: *International Conference on Applied Cryptography and Network Security*. Springer.

- 2006, pp. 98–113. URL: <https://pdfs.semanticscholar.org/7f4c/79649a55eecd8c2fbf4f8b0c5e68e2376a.pdf> (cit. on p. 36). [43]
- [33] Cynthia Dwork, Andrew Goldberg, and Moni Naor. “On memory-bound functions for fighting spam”. In: *Annual International Cryptology Conference*. Springer. 2003, pp. 426–444. URL: <http://www.wisdom.weizmann.ac.il/mathusers/naor/PAPERS/mem.pdf> (cit. on p. 36).
- [34] Morris Dworkin. *Recommendation for block cipher modes of operation. methods and techniques*. Tech. rep. DTIC Document, 2001 (cit. on p. 24).
- [35] M. Elkins. *RFC2015 MIME Security with Pretty Good Privacy (PGP)*. IETF, 1996. URL: <http://tools.ietf.org/pdf/rfc2015.pdf> (cit. on p. 1).
- [36] *Email Client Market Share*. 2014. URL: <http://emailclientmarketshare.com/> (cit. on p. 33).
- [37] “Ethash”. In: (2013). URL: <https://github.com/ethereum/wiki/wiki/Ethash> (cit. on p. 35).
- [38] Nathan Evans, Roger Dingledine, and Christian Grothoff. “A Practical Congestion Attack on Tor Using Long Paths”. In: *Proceedings of the 18th USENIX Security Symposium*. Aug. 2009. URL: <http://freehaven.net/anonbib/papers/congestion-longpaths.pdf> (cit. on p. 28).
- [39] *Fact Sheet Suite B Cryptography*. 2008. URL: http://www.nsa.gov/ia/industry/crypto_suite_b.cfm (cit. on p. 23). *FIXME missing document link*
- [40] W. Feng, E. Kaiser, and A. Luu. “Design and implementation of network puzzles”. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 4. Mar. 2005, 2372–2382 vol. 4. DOI: 10.1109/INFCOM.2005.1498523. URL: <http://ieeexplore.ieee.org/document/1498523/> (cit. on p. 35).
- [41] W. c. Feng and E. Kaiser. “The Case for Public Work”. In: *2007 IEEE Global Internet Symposium*. May 2007, pp. 43–48. DOI: 10.1109/GI.2007.4301429 (cit. on p. 35).
- [42] Christian Forler, Stefan Lucks, and Jakob Wenzel. “Memory-demanding password scrambling”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2014, pp. 289–305. DOI: 10.1007/978-3-662-45608-8_16. URL: http://link.springer.com/chapter/10.1007/978-3-662-45608-8_16 (cit. on p. 36).
- [43] N. Freed and N. Borenstein. *RFC2045 Multipurpose Internet Mail Extensions; (MIME) Part One: Format of Internet Message Bodies*. IETF, 1996. URL: <http://tools.ietf.org/pdf/rfc2045.pdf> (cit. on pp. 1, 17).
- [44] N. Freed and N. Borenstein. *RFC2046 Multipurpose Internet Mail Extensions; (MIME) Part Two: Media Types*. IETF, 1996. URL: <http://tools.ietf.org/pdf/rfc2046.pdf> (cit. on p. 17).
- [45] Praveen Gauravaram, Lars R Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S Thomsen. “Grøstl-a SHA-3 candidate”. In: *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2009 (cit. on p. 35).
- [46] R. Gellens and J. Klensin. *RFC4409 Message Submission for Mail*. IETF, 2006. URL: <http://tools.ietf.org/pdf/rfc4409.pdf> (cit. on p. 32).
- [47] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC ’09. Bethesda, MD, USA: ACM, 2009, pp. 169–178. ISBN: 978-1-60558-506-2. DOI: 10.1145/1536414.1536440. URL: <http://doi.acm.org/10.1145/1536414.1536440> (cit. on p. 23).
- [48] Sharad Goel, Mark Robson, Milo Polte, and Emin Gun Sirer. *Herbivore: A Scalable and Efficient Protocol for Anonymous Communication*. Tech. rep. 2003-1890. Ithaca, NY: Cornell University, Feb. 2003. URL: <http://www.cs.cornell.edu/People/egs/papers/herbivore-tr.pdf> (cit. on p. 30).
- [49] Philippe Golle and Ari Juels. “Dining Cryptographers Revisited”. In: *Proceedings of Eurocrypt 2004*. May 2004. URL: <http://crypto.stanford.edu/~pgolle/papers/nim.pdf> (cit. on p. 27).
- [50] Bogdan Groza and Dorina Petrica. “On Chained Cryptographic Puzzles”. In: URL: https://www.researchgate.net/profile/Dorina_Petrica/publication/228973617_On_chained_cryptographic_puzzles/_links/004635310530eb4340000000.pdf (cit. on p. 35).

- [51] Bogdan Groza and Dorina Petrica. "Cryptanalysis of an authentication protocol". In: *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05)*. IEEE. 2005, 7–pp. URL: https://www.researchgate.net/profile/Dorina_Petrica/publication/4223450_Cryptanalysis_of_an_authentication_protocol/links/004635310530f14ff7000000.pdf (cit. on p. 36). 
- [52] Bogdan Groza and Bogdan Warinschi. "Revisiting difficulty notions for client puzzles and DoS resilience". In: *International Conference on Information Security*. Springer. 2012, pp. 39–54. DOI: 10.1.1.716.9775. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.716.9775&rep=rep1&type=pdf> (cit. on p. 36). 
- [53] Ceki Gülcü and Gene Tsudik. "Mixing E-mail With Babel". In: *Proceedings of the Network and Distributed Security Symposium - NDSS '96*. IEEE, Feb. 1996, pp. 2–16. URL: <http://citeseer.nj.nec.com/2254.html> (cit. on p. 28). 
- [54] Michael Herrmann and Christian Grothoff. "Privacy Implications of Performance-Based Peer Selection by Onion Routers: A Real-World Case Study using I2P". In: *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS 2011)*. Waterloo, Canada, July 2011. URL: <http://freehaven.net/anonbib/papers/pets2011/p9-herrmann.pdf> (cit. on p. 29). 
- [55] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. "The Parrot is Dead: Observing Unobservable Network Communications". In: *Proceedings of the 2013 IEEE Symposium on Security and Privacy*. May 2013. URL: <http://www.cs.utexas.edu/~amir/papers/parrot.pdf> (cit. on pp. 21, 81). 
- [56] Jan Iwanik, Marek Klonowski, and Miroslaw Kutylowski. "DUO-Onions and Hydra-Onions—failure and adversary resistant onion protocols". In: *Communications and Multimedia Security*. Springer. 2005, pp. 1–15. URL: <http://dl.ifip.org/db/conf/cms/cms2004/IwanikKK04.pdf> (cit. on p. 30). 
- [57] AJP Jeckmans. "Practical client puzzle from repeated squaring". In: (2009). URL: http://essay.utwente.nl/59133/1/scriptie_A_Jeckmans.pdf (cit. on p. 34). FIXME missing document link
- [58] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries". In: *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)*. Nov. 2013. URL: <http://www.ohmygodel.com/publications/usersrouted-ccs13.pdf> (cit. on p. 29). 
- [59] Simon Josefsson and Colin Percival. "RFC7914: The scrypt Password-Based Key Derivation Function". In: (2016). URL: <https://tools.ietf.org/pdf/rfc7914.pdf> (cit. on p. 36). 
- [60] Ari Juels and John G Brainard. "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks". In: *NDSS*. Vol. 99. 1999, pp. 151–165. URL: <http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/juels.pdf> (cit. on pp. 34, 35). 
- [61] J. Klensin. *RFC5321 Simple Mail Transfer Protocol*. IETF, 2008. URL: <http://tools.ietf.org/pdf/rfc5321.pdf> (cit. on pp. 17, 31, A52). 
- [62] Neal Koblitz, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. 2004. URL: <http://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf> (cit. on p. 23). 
- [63] Butler W. Lampson. *A Note on the Confinement Problem*. 1973. URL: <http://faculty.kfupm.edu.sa/COE/mimam/Papers/73%20A%20Note%20on%20the%20Confinement%20Problem.pdf> (cit. on p. 21). 
- [64] Arjen K. Lenstra. *Key Length. Contribution to The Handbook of Information Security*. 2004. URL: <https://infoscience.epfl.ch/record/164539/files/NPDF-32.pdf> (cit. on p. 23). 
- [65] Brian Neil Levine and Clay Shields. "Hordes — A Multicast Based Protocol for Anonymity". In: *Journal of Computer Security* 10.3 (2002), pp. 213–240. URL: <http://prisms.cs.umass.edu/brian/pubs/brian.hordes.jcs01.pdf> (cit. on p. 30). 
- [66] Helger Lipmaa, Phillip Rogaway, and David Wagner. "CTR-mode encryption". In: *First NIST Workshop on Modes of Operation*. 2000 (cit. on p. 24). 

- [67] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. "l-diversity: Privacy beyond k-anonymity". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), p. 3. URL: <http://www.cs.cornell.edu/~vmuthu/research/ldiversity.pdf> (cit. on p. 19). 
- [68] Mitsuru Matsui, S Moriai, and J Nakajima. "RFC3713: A Description of the Camellia Encryption Algorithm". In: (2004) (cit. on p. 22). 
- [69] David McGrew and John Viega. "The Galois/counter mode of operation (GCM)". In: *Submission to NIST*. <http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf> (2004) (cit. on p. 25). 
- [70] David A McGrew and John Viega. "The security and performance of the Galois/Counter Mode (GCM) of operation". In: *International Conference on Cryptology in India*. Springer. 2004, pp. 343–355 (cit. on p. 25). 
- [71] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Ed. by Hugh C. Williams. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426. ISBN: 978-3-540-39799-1. DOI: 10.1007/3-540-39799-X_31. URL: http://dx.doi.org/10.1007/3-540-39799-X_31 (cit. on p. 23).
- [72] Kazuhiko Minematsu, Stefan Lucks, Hiraku Morita, and Tetsu Iwata. "Attacks and security proofs of EAX-prime". In: *International Workshop on Fast Software Encryption*. Springer. 2013, pp. 327–347. URL: <http://eprint.iacr.org/2012/018.pdf> (cit. on p. 24). 
- [73] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S Wallach. "AP3: Cooperative, decentralized anonymous communication". In: *Proceedings of the 11th workshop on ACM SIGOPS European workshop*. ACM. 2004, p. 30. URL: <http://www-dev.ccs.neu.edu/home/amislove/publications/AP3-SIGOPSEW.pdf> (cit. on p. 30). 
- [74] Prateek Mittal and Nikita Borisov. "Information Leaks in Structured Peer-to-peer Anonymous Communication Systems". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*. Ed. by Paul Syverson, Somesh Jha, and Xiaolan Zhang. Alexandria, Virginia, USA: ACM Press, Oct. 2008, pp. 267–278. URL: <http://www.hatswitch.org/~nikita/papers/>
- [75] Jerey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. DOI: 10.17487/rfc2616. URL: <https://rfc-editor.org/rfc/rfc2616.txt> (cit. on p. 15). FIXME missing document link
- [76] Arjun Nambiar and Matthew Wright. "Salsa: A Structured Approach to Large-Scale Anonymity". In: *Proceedings of CCS 2006*. Nov. 2006. URL: <http://ranger.uta.edu/~mwright/papers/salsa-ccs06.pdf> (cit. on p. 30). FIXME missing document link
- [77] Suresh Venkatasubramanian Ninghui Li Tiancheng Li. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity". In: (). URL: http://www.cs.purdue.edu/homes/li83/papers/icde_closeness.pdf (cit. on p. 19). 
- [78] Lasse Øverlier and Paul Syverson. "Locating Hidden Servers". In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006. URL: <http://tor-svn.freehaven.net/anonbib/cache/hs-attack06.pdf> (cit. on p. 28). FIXME missing document link
- [79] Colin Percival. "Stronger key derivation via sequential memory-hard functions". In: *Self-published* (2009), pp. 1–16 (cit. on p. 36). 
- [80] Alexander Peslyak. "yescrypt - a Password Hashing Competition submission". In: Mar. 2014. URL: https://password-hashing.net/wiki/doku.php/yescrypt#download_submission (cit. on p. 35). 
- [81] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf. v0.34. Aug. 2010. URL: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf (cit. on pp. 19, 20). FIXME missing document link
- [82] Jon Postel and Joyce Reynolds. *RFC 959: File transfer protocol*. 1985. URL: <https://tools.ietf.org/pdf/rfc959.pdf> (cit. on p. 15).

- [83] B. Ramsdell. *RFC2440 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*. IETF, 2004. URL: <http://tools.ietf.org/pdf/rfc2440.pdf> (cit. on p. 33). 
- [84] B. Ramsdell. *RFC3851 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*. IETF, 2004. URL: <http://tools.ietf.org/pdf/rfc3851.pdf> (cit. on p. 33). 
- [85] Marc Rennhard and Bernhard Plattner. "Practical Anonymity for the Masses with Mix-Networks". In: *Proceedings of the IEEE 8th Intl. Workshop on Enterprise Security (WET ICE 2003)*. Linz, Austria, June 2003. URL: <https://gnunet.org/sites/default/files/RP03-1.pdf> (cit. on p. 26). FIXME missing document link
- [86] P. Resnick. *RFC5322 Internet Message Format*. IETF, 2008. URL: <http://tools.ietf.org/pdf/rfc5322.pdf> (cit. on p. 17). 
- [87] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: <http://doi.acm.org/10.1145/359340.359342> (cit. on p. 23). FIXME missing document link
- [88] Ronald L Rivest, Adi Shamir, and David A Wagner. "Time-lock puzzles and timed-release crypto". In: (1996). URL: <http://bitsavers.trailing-edge.com/pdf/mit/lcs/tr/MIT-LCS-TR-684.pdf> (cit. on p. 35). 
- [89] Phillip Rogaway, Mihir Bellare, and John Black. "OCB: A block-cipher mode of operation for efficient authenticated encryption". In: *ACM Transactions on Information and System Security (TISSEC)* 6.3 (2003), pp. 365–403 (cit. on p. 24). 
- [90] Phillip Rogaway and Ted Krovetz. *The OCB Authenticated-Encryption Algorithm*. Internet-Draft draft-krovetz-ocb-04. Work in Progress. Internet Engineering Task Force, July 2012. 17 pp. URL: <https://tools.ietf.org/html/draft-krovetz-ocb-04> (cit. on p. 24). 
- [91] J. P. Sain-Andre. *RFC3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)*. IETF, 2004. URL: <http://tools.ietf.org/pdf/rfc3923.pdf> (cit. on p. 17). 
- [92] P. Saint-Andre. *RFC3922: Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)*. IETF, 2004. URL: <http://tools.ietf.org/pdf/rfc3922.pdf> (cit. on p. 17). 
- [93] P. Saint-Andre. *RFC6120: Extensible Messaging and Presence Protocol (XMPP): Core*. IETF, 2011. URL: <http://tools.ietf.org/pdf/rfc6120.pdf> (cit. on p. 17). 
- [94] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. "P5: A Protocol for Scalable Anonymous Communication". In: *Proceedings of the 2002 IEEE Symposium on Security and Privacy*. May 2002. URL: <http://www.cs.umd.edu/projects/p5/p5.pdf> (cit. on p. 30). FIXME missing document link
- [95] Peter W. Shor. "Polynomial-Time Algorithms For Prime Factorization And Discrete Logarithms On A Quantum Computer". In: *SIAM Journal on Computing* 26 (1997), pp. 1484–1509. URL: <https://arxiv.org/pdf/quant-ph/9508027v2.pdf> (cit. on p. 23). FIXME missing document link
- [96] NIST-FIPS Standard. "Announcing the advanced encryption standard (AES)". In: *Federal Information Processing Standards Publication* 197 (2001), pp. 1–51 (cit. on p. 22). 
- [97] A. Stevenson. *Oxford Dictionary of English*. Oxford reference online premium. OUP Oxford, 2010. ISBN: 9780199571123. URL: <http://www.oed.com> (cit. on p. A51).
- [98] Almon Brown Strowger. "Automatic Telephone-Exchange". en. Pat. 447918. Mar. 1891 (cit. on p. 1). FIXME missing document link
- [99] Mansi S Subhedar and Vijay H Mankar. "Current status and key issues in image steganography: A survey". In: *Computer science review* 13 (2014), pp. 95–113 (cit. on p. 22). 
- [100] Biaoshuai Tao and Hongjun Wu. "Improving the biclique cryptanalysis of AES". In: *Australasian Conference on Information Security and Privacy*. Springer, 2015, pp. 39–56 (cit. on p. 22). 
- [101] "The Lyra2 reference guide". In: 2015. URL: <https://password-hashing.net/wiki/doku.php/lyra2> (cit. on p. 35). FIXME missing document link

- [102] Muldowney Thomas, Miller Mathew, Eatmon Ryan, and Saint-Andre Peter. *XEP-0096: SI File Transfer*. XMPP Standards Foundation, 2004. URL: <http://xmpp.org/extensions/xep-0096.html> (cit. on p. 17). FIXME missing document link
- [103] Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto. "Toward non-parallelizable client puzzles". In: *International Conference on Cryptology and Network Security*. Springer. 2007, pp. 247–264. DOI: 10.1007/978-3-540-76969-9_16. URL: http://link.springer.com/chapter/10.1007/978-3-540-76969-9_16 (cit. on p. 36). 
- [104] John Tromp. "Cuckoo Cycle: a memory bound graph-theoretic proof-of-work". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2015, pp. 49–62 (cit. on p. 36). 
- [105] UNHR. *International Covenant on Civil and Political Rights*. 1966. URL: <http://www.ohchr.org/en/professionalinterest/pages/ccpr.aspx> (cit. on p. 11). FIXME missing document link
- [106] Andrew J Viterbi. *CDMA: principles of spread spectrum communication*. Addison Wesley Longman Publishing Co., Inc., 1995 (cit. on p. 22).
- [107] Michael Waidner and Birgit Pfitzmann. "The dining cryptographers in the disco: Unconditional Sender and Recipient Untraceability". In: *Proceedings of EUROCRYPT 1989*. Springer-Verlag, LNCS 434, 1990. URL: http://www.semper.org/sirene/publ/WaPf1_89DiscoEngl.ps.gz (cit. on p. 27). FIXME missing document link
- [108] Brent Waters, Ari Juels, J Alex Halderman, and Edward W Felten. "New client puzzle outsourcing techniques for DoS resistance". In: *Proceedings of the 11th ACM conference on Computer and communications security*. ACM. 2004, pp. 246–256. URL: http://www.cs.utexas.edu/~bwaters/publications/papers/outsource_paper.pdf (cit. on p. 35). 
- [109] Doug Whiting, Niels Ferguson, and Russell Housley. "RFC3610: Counter with cbc-mac (ccm)". In: (2003) (cit. on p. 25). 
- [110] Wikipedia. *Steganographie*. 2013. URL: <https://en.wikipedia.org/wiki/Steganographie> (cit. on p. 22). 
- [111] *Windows GPU Miners for the More Commonly Used Crypto Algorithms*. Cryptomining Blog. June 2014. URL: <http://cryptomining-blog.com/2595-windows-gpu-miners-for-the-more-commonly-used-crypto-algorithms/> (cit. on p. 34).
- [112] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. "Dissent in numbers: Making strong anonymity scale". In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 2012, pp. 179–182 (cit. on p. 30). 
- [113] Li Zhuang, Feng Zhou, Ben Y Zhao, and Antony Rowstron. "Cashmere: Resilient anonymous routing". In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-VOLUME 2*. USENIX Association. 2005, pp. 301–314 (cit. on p. 31). FIXME missing document link

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 post-graduate study 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 constantly broadened his horizon by working for industry, banking and government as 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 main expertise lays in the field of networking related problems dealing with data protection, distribution, confidentiality and anonymity.

