



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

November 12, 2016

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, FIXME datum der fakultätsversammlung FIXME Signature line with Name of Dean of Faculty below

Contents

1. Introduction	1
1.1. Contributions	2
1.2. Notation	2
1.2.1. Cryptography	2
1.2.2. Code and commands	2
1.2.3. Hyperlinking	2
2. Main Research Question	3
2.1. RQ1: Sending messages maintaining unlinkability against an adversary	3
2.2. RQ2: Attacking unlinkability and circumvention	3
2.3. RQ3: Mitigation by design	3
3. Methodes	5
4. Requirements for an anonymising Protocol	9
3.1. Anonymizing and unlinking	9
3.2. Censorship Resistant	9
3.3. Controllable trust	10
3.4. Reliable	10
3.5. Diagnosable	10
3.6. Available	10
3.7. Rough draft of Protocol Layers	10
3.8. Rough Draft of Messages	11
5. Existing Transport Layer Protocols	13
4.1. HTTP	13
4.2. MQTT	13
4.3. Advanced Message Queuing Protocol	13
4.4. Constrained Application Protocol (CoAP)	14
4.5. Web Application Messaging Protocol	14
4.6. XMPP (jabber)	14
4.7. SMTP	14
4.8. SMS	15
4.9. MMS	15
4.10. Roundup for Transport Protocols	15
6. Existing Research and Implementations on the Topic	17
5.1. Anonymity	17
5.1.1. k -Anonymity	17
5.1.2. ℓ -Diversity	17
5.1.3. t -Closeness	17
5.2. Zero Trust	17
5.3. Pseudonymity	18
5.4. Undetectability	18
5.5. Unobservability	18
5.5.1. Ephemeral Identity	18
5.6. Single Use Reply Blocks and Multi Use Reply Blocks	18
5.7. Censorship	18
5.7.1. Censorship Resistant	19
5.7.2. Parrot Circumvention	19
5.7.3. Censorship Circumvention	19
5.7.3.1. Covert Channel	19
5.7.3.2. Spread Spectrum	19
5.8. Cryptography	19
5.8.1. Symmetric Encryption	19
5.8.1.1. AES	19
5.8.2. Asymmetric Encryption	19
5.8.2.1. RSA	19

5.8.2.2.	Elliptic Curve Cryptography	20
5.8.3.	Homomorphic encryption	20
5.8.4.	Deniable Encryption and Deniable Steganography	20
5.8.5.	Key Sizes	20
5.9.	Routing	20
5.9.1.	Mixing	20
5.9.2.	Onion Routing	21
5.9.3.	Crowds	21
5.9.4.	Mimic routes	21
5.9.4.1.	DC Networks	21
5.9.4.2.	Anonymous Remailer	21
5.9.5.	Cryptopuzzles	22
5.9.5.1.	Argon2	22
5.10.	System Implementations	22
5.10.1.	Pseudonymous Remailer	22
5.10.2.	I^2P	22
5.10.3.	Babel	22
5.10.4.	Cypherpunk-Remailer	22
5.10.5.	Mixmaster-Remailer	22
5.10.6.	Mixminion-Remailer	22
5.10.7.	Crowds	22
5.10.8.	Tarzan	23
5.10.9.	Tor	23
5.10.10.	Herbivore	23
5.10.11.	Dissent	23
5.10.12.	P5	23
5.10.13.	Gnutella	23
5.10.14.	Gnutella2	23
5.10.15.	Freenet	23
5.10.16.	Darknet	23
5.10.17.	Sneakernet	23
5.10.18.	Hordes	23
5.10.19.	Salsa	23
5.10.20.	Hydra-Onion	23
5.10.21.	SMTP	23
5.10.22.	Mail Endpoints	24
5.10.22.1.	Fat clients	25
5.10.22.2.	Server located clients	25
5.10.22.3.	Web clients	25
5.10.23.	Interfaces of Mail Endpoints	25
5.10.24.	S/MIME	25
5.10.25.	PGP/MIME	25
5.10.26.	XMMP	25
5.11.	Known Attacks	25
5.11.1.	Broken Encryption Algorithms	25
5.11.2.	Attacks Targeting Anonymity	25
5.11.2.1.	Hotspot Attacks	25
5.11.2.2.	Message Tagging and Tracing	25
5.11.2.3.	Side Channel Attacks	26
5.11.2.4.	Timing Attacks	26
5.11.2.5.	Sizing Attacks	26
5.11.2.6.	Bugging Attacks	26
5.11.3.	Denial of Service Attacks	26
5.11.3.1.	Censorship	26
5.11.3.2.	Credibility Attack	26
6.	Applied Methodes	27
6.1.	Problem Hotspots	27
6.1.1.	Zero Trust Philosophy	27
6.1.2.	Information leakage	27
6.1.2.1.	P2P design	27
6.1.2.2.	Decoy traffic generation	27

6.1.2.3.	Message tagging or bugging protection	28
6.1.2.4.	Message replay protection	28
6.1.2.5.	No Dedicated Infrastructure Philosophy	28
6.1.3.	Accounting	28
6.1.4.	Anonymisation	28
6.1.5.	Initial Bootstrapping	28
6.1.6.	Cypher selection	28
6.1.7.	Usability	29
6.2.	Protocol Specification	29
6.2.1.	MURBs	29
6.3.	Protocol implementation	29
6.3.1.	Header block	29
6.3.1.1.	Ephemeral Identity	29
6.3.1.2.	Requests	29
6.3.1.3.	Replies	29
6.3.1.4.	Crypto Puzzle	29
6.3.2.	Main block	29
6.3.2.1.	Routing Blocks	29
6.3.2.2.	Routing Log Block	29
6.3.2.3.	Reply Block	29
6.3.2.4.	payload Block	29
6.4.	Protocol Analysis	29
II.	Results	31
7.	MessageVortex - Transport Independent Messaging anonymous to 3 rd Parties	35
7.1.	Protocol Description	35
7.2.	Accounting	35
7.3.	Message Flows	35
7.4.	Considerations for Building Messages	35
7.4.1.	Ephemeral identities	35
7.4.2.	Timing of messages	35
7.4.3.	Building Diagnostic Paths	36
7.4.3.1.	Implicit Diagnostic	36
7.4.3.2.	Automatic Explicit Diagnostic	36
7.4.3.3.	On-Demand Explicit Diagnostic	36
7.5.	Real World Considerations	36
8.	Security Analysis	37
9.	Additional Considerations	39
9.1.	Storage of Messages and queues	39
9.2.	Economy of transfer	39
III.	Discussion	41
10.	Anonymity	45
10.1.	Effects of anonymous communication on behaviour	45
IV.	Appendix	47
ASN.1	representation of the protocol	49
Glossary	53
Bibliography	55

List of Tables

4.1. comparison of protocols in terms of the suitability criteria as transport layer 15

List of Figures

3.1. A traceroute to the host www.ietf.org	9
3.2. A first rough overview for the protocol	10
5.1. Mail Agents	24
7.1. Overview of the Vortex modules	35

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" [38]. 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 have shown that data is broadly collected 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 (covering roughly 10% of the natural and juristic entities within Switzerland at that time).

A series of similar attempts to attack privacy on a global scale have been discovered by whistle blower 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 – due to the fact that 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 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 germans 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[13] or PGP[10]) 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 has to be avoided.

In this work a new protocol is designed to allow message transfer through existing communication channels. These messages should be unobservable to any third party. This unobservability does not only cover the message itself but all metadata and flows associated with it. The protocol should be designed in such a way so that it is capable to use any type of transfer

1. Introduction

protocol. It should be even possible to switch protocols while have messages in transfer to allow media breaches (at least on a protocol level).

The new protocol should allow safe communication without the need of trusting the underlying transport media. Furthermore it is desirable that 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 are collecting 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. Try to elaborate to what extend the protocol fulfils the requirements of this work.

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. Messages encrypted with multiple keys do list the used keys as a coma 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 G^{K_a} where a is an index to distinguish different keys. Decryption uses therefore $D^{K_a}(M^{K_a})$.

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 notation $M(\text{timebase}, \text{MURB}, \text{serialNumber})$ respectively if encrypted $M_{K_a}(\text{timebase}, \text{MURB}, \text{serialNumber})$.

$$\begin{aligned} \text{asymmetric} : E^{K_a^{-1}}(M) &= M^{K_a^{-1}} \\ D^{K_a^1}(E^{K_a^{-1}}(M)) &= D^{K_a^{-1}}(E^{K_a^1}(M)) = M \\ \text{symetric} : E^{K_a}(M) &= M^{K_a} \\ D^{K_a}(E^{K_a}(M)) &= M \\ \text{hashing(unsalted)} : H(M) &= H_M \\ \text{hashing(salted)} : H^{S_a}(M) &= H_M^{S_a} \end{aligned}$$

In general Subscripts denote selectors to differentiate values of the same type and superscript denote relevant parameters to operations expressed.

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.out.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 "↵" is inserted to indicate that the line break has been introduced for readability.

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

1.2.3. Hyperlinking

The electronic version of this document is hyperlinked. 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.

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? (RQ1)
2. How can entities utilizing these technologies and methods be attacked and what measures are available to circumvent such attacks? (RQ2)
3. How can attacks targeting anonymity of a sending or receiving entity be mitigated by design? (RQ3)

2.3. RQ3: 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 RQ1.

2.1. RQ1: Sending messages maintaining unlinkability against an adversary

This question covers the principal part of the work. We try to elaborate first rough criterias for the 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. RQ2: 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).

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.

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)[40].

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), Qatar, 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 through 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 being sent be seen 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 extent.

```
1 $ traceroute www.ietf.org
2 traceroute to www.ietf.org.cdn.cloudflare-
  dnssec.net (104.20.0.85), 64 hops max
3 1 147.86.8.253 0.418ms 0.593ms 0.421ms
4 2 10.19.0.253 1.177ms 0.829ms 0.782ms
5 3 10.19.0.253 0.620ms 0.427ms 0.402ms
6 4 193.73.125.35 1.121ms 0.828ms 0.905ms
7 5 193.73.125.81 2.991ms 2.450ms 2.414ms
8 6 193.73.125.81 2.264ms 1.961ms 1.959ms
9 7 192.43.192.196 6.472ms 199.543ms
  201.152ms
10 8 130.59.37.105 3.465ms 3.138ms 3.121ms
11 9 130.59.36.34 3.904ms 3.897ms 4.989ms
12 10 130.59.38.110 3.625ms 3.333ms 3.379ms
13 11 130.59.36.93 7.518ms 7.232ms 7.246ms
14 12 130.59.38.82 7.155ms 17.166ms 7.034ms
15 13 80.249.211.140 22.749ms 22.415ms
  22.467ms
16 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 ?? 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:

3. Requirements for an anonymising Protocol

- 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 jurisdiction's 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 5.10.9) 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

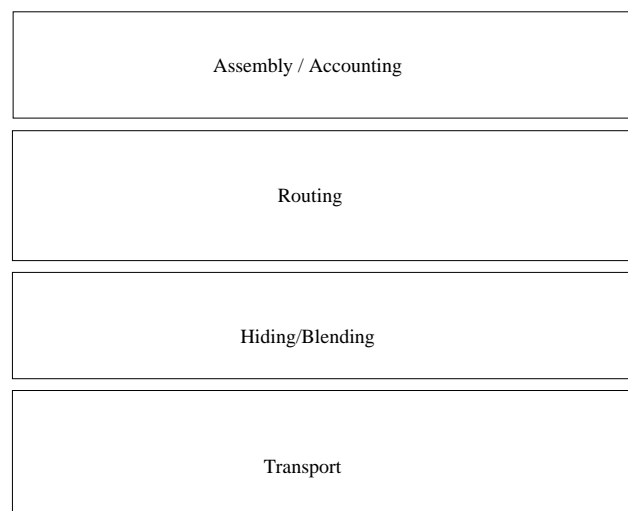


Figure 3.2.: A first rough overview for the protocol

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

3. Requirements for an anonymising Protocol

messages.

- Routing Blocks
This block contains information of which data block should be sent to what recipient. 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 protocols. 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 do 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.

4.1. HTTP

The HTTP protocol allows message transfer from and to a server and is specified in RFC2616 [24]. 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. 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 [4]) 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.

4.2. MQTT

MQTT is an ISO standard (ISO/IEC PRF 20922:2016) and was formerly called MQ Telemetry Transport. The current standard as the time of writing this document was 3.1.1 [3].

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 enduser (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. 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[2].

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[2].

4. Existing Transport Layer Protocols

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).

of security including end-to-end signing and object encryption[33]. There is also a stream initiation extension for transferring files between endpoints [39].

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.



4.4. 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[5]. 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.

4.5. 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.

4.6. XMPP (jabber)

XMPP (originally named Jabber) is a synchronous message protocol used in the internet. It is specified in the documents RFC6120[35], RFC6121[35], RFC3922[34], and RFC3923[33]. The protocol is a very advanced chat protocol featuring numerous levels

4.7. SMTP

The SMTP protocol is currently specified in [19]. 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 [31] and is separated with a blank line from the body.

It [19] furthermore introduces a simplistic model for smtp message based communication. A more comprehensive model is introduced in the section ???. 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 [13], and [14]. 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) 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 and receiving messages. Messages going over several servers are common (Ct3: Symmet-

rically built) so the third criteria may be considered as fulfilled as well

SMTP is considered a strong candidate as transport layer.

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).

4.8. 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.



4.9. MMS

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



4.10. Roundup for Transport Protocols

Criteria \ Protocol	Ct1: Widely adopted	Ct2: Reliable	Ct3: Symmetrically built
HTTP	✓	✓	×
MOTT	~	✓	×
AMQP	~	✓	×
CoAP	~	✓	×
WAMP	×	✓	~
XMPP	✓	✓	✓
SMTP	✓	✓	✓
SMS ¹	✓	✓	×
MMS	✓	✓	×

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

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 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

¹ omitted 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 [26].

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 [1]. 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*



5.1.2. ℓ -Diversity

In [22] an extended model of k -anonymity is introduced. In this paper the 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[25] 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.

¹footnotes omitted in quote

5.3. Pseudonymity

As Pseudonymity we take the definition as specified in [26].

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 [26].

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 [26].

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 (\Rightarrow reads "implies"):

censorship resistance \Rightarrow *unobservability*
unobservability \Rightarrow *undetectability*
unobservability \Rightarrow *anonymity*

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".

²footnotes omitted in quote

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 [7]. 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 Blocks" (SURBs) which may be used once and "Multi Use Reply Blocks" (MURBs) which may be used a limited number of times.

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 re-

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 [18] Houmansadr, 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



5.7.3.2. Spread Spectrum



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

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

For a key $K_b \neq K_a$ this means

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

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

5.8.1.1. AES



5.8.2. Asymmetric Encryption

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

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

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

It is important that

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

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

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

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

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

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

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

5.8.2.1. RSA

In 1978 the authors Rivest, Shamir, and Adleman published with [32] a paper which did revolutionize cryptography for years. In their paper the authors described an encryption method later to be called RSA

which required a key pair (K_a) referenced as public (K_a^1) and private keys (K_a^{-1}). The novelty of this system was that anything encrypted with the public key was only decryptable with the private key and vice versa.

RSA is up until the day of writing this paper not publicly known to be broken (unless a too small key size is used). However – Shor described in 1997 an algorithm which should enable quantum computers to break RSA far faster than done with common computers. In the section 5.8.5 we do elaborate this effects further.

5.8.2.2. Elliptic Curve Cryptography

The elliptic curves were independently suggested by [23] and [20] 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



5.8.4. Deniable Encryption and Deniable Steganography



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 [21] 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 [12] 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.

5.9. 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”[7] 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 suitability 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 adverser.

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 adversers 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 5.10.9.



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 Jhondo 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 jondos.



5.9.4. Mimic routes

Mimics are a set of static 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[8]. 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[16][41]. 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” seen  have more than one implementation in the wild. ■

Pseudonymous Remailers (also called Nym Servers) take a message and replace all information pointing to the original sender with a pseudonym. This

pseudonym may be used as an answer address. The most well known pseudonymous remailer possibly was anon.penet.fi run by Johan Helsingius. This service has been forced several times to reveal a pseudonyms true identity before Johan Helsingius decided to shut it down. For a more in depth discussion of Pseudonymous Remainers see ??

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 Remainers see 5.10.4.

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 5.10.5.

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

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

For a more in depth discussion of Mixminion remailers see 5.10.6.

5.9.5. Cryptopuzzles



5.9.5.1. Argon2



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 [7]. The most well known remailer was probably anon.penet.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. I^2P



5.10.3. Babel

Babel was an academic system defined in  paper by Gülcü and Tsudik in 1996[17].

5.10.4. Cypherpunk-Remailer



5.10.5. Mixmaster-Remailer



5.10.6. Mixminion-Remailer



5.10.7. Crowds



5.10.8. Tarzan**5.10.9. Tor****5.10.10. Herbivore****5.10.11. Dissent****5.10.12. P5****5.10.13. Gnutella****5.10.14. Gnutella2****5.10.15. Freenet****5.10.16. Darknet****5.10.17. Sneakernet****5.10.18. Hordes****5.10.19. Salsa****5.10.20. Hydra-Onion****5.10.21. SMTP**

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

Today's mail transport is mostly done via SMTP protocol as specified in [19]. 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 (whereas the latter is usually implemented by a push-pull mechanism).

To understand the routing of a mail it is essential to understand the whole chain starting from a user(-agent) until arriving at the target user (and being read!). To simplify this 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.

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" is 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).

5. Existing Research and Implementations on the Topic

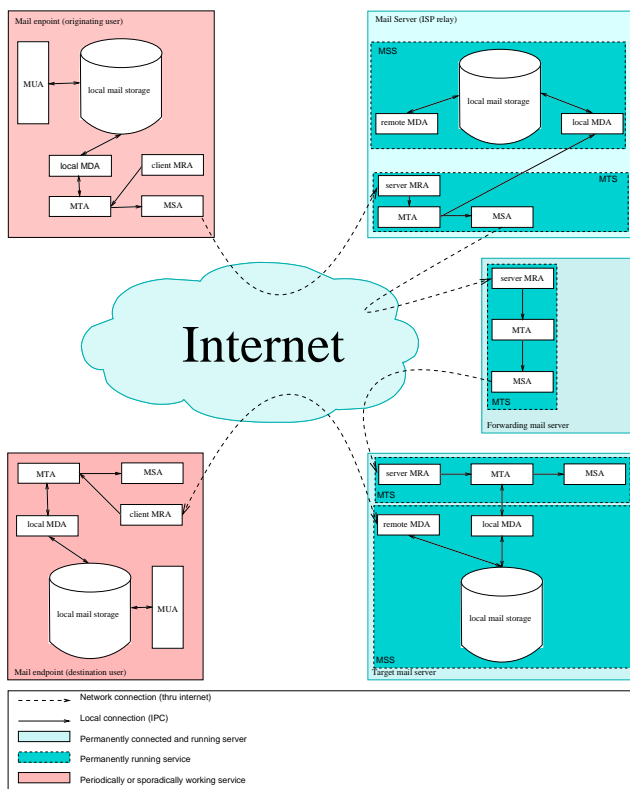


Figure 5.1.: Mail Agents

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.

5.10.22. 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 [15]).
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/passsword or certificate) or data identifying the sending system (eg. IP address)[15]. Failure in doing authentication may result in this port beeing misused as an 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 im-

plemented as a database in a user specific directory structure.



5.10.22.1. 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 (thru 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 on-line. 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 thru 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 [6] 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*[11] as a more recent source lists in February 2014 iOS devices with 37%, followed by Outlook (13%), and Google Android (9%).

5.10.22.2. 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.

5.10.24. S/MIME



5.10.25. PGP/MIME



5.10.26. XMPP



5.11. 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.

5.11.1. 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 threats in a timely manner. This reaction should not rely on a required update of the infrastructure.



5.10.22.3. 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 [11] the most common web clients are "Gmail", "Outlook.com", and "Yahoo! Mail". All these Interfaces do not offer a kind of public plugin interface. However, they do offer IMAP-interfaces. This important for a future generalistic approach to the problem.

5.11.2. Attacks Targeting Anonymity



5.11.2.1. Hotspot Attacks



5.10.23. Interfaces of Mail Endpoints

There are two interfaces

5.11.2.2. Message Tagging and Tracing



5. Existing Research and Implementations on the Topic

5.11.2.3. Side Channel Attacks



5.11.2.4. Timing Attacks



5.11.2.5. Sizing Attacks



5.11.2.6. Bugging Attacks



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.

5.11.3. Denial of Service Attacks

5.11.3.1. 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.

5.11.3.2. 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

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

6.1.2. 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.

The message should be untaggable (neither by a sender, or by an intermediate party such as a mixer) and unbuggable.

In order to make a message untraceable it should be able to increase and shrink in size or all messages must have uniform size. Decoy traffic should be distinguishable from true message traffic.

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.

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.

6.1.2.2. Decoy traffic generation

In order to create decoy traffic in an untrusted way we need means to increase and 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 however 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 result the system would be suitable to blackmail any user of the world. It furthermore would create unintentional “exit nodes” to the system which might oppose further legal threads for participants.

6.1.2.4. Message replay protection

Message replay 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.

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.

6.1.3. Accounting



6.1.4. Anonymisation



6.1.5. Initial Bootstrapping



6.1.6. Cypher selection

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

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 necessary and (b) there is an alternative if an algorithm 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 widely common in implementations they have already been standardized as they build a possibility to step up security in future.

Having this criterias for choice I chose to use the following keys and key sizes:

- Symetric
 - AES (Keysizes 128, 192, 256)
 - Camellia (Keysizes 128, 192, 256)
- Asymetric
 - RSA/DSA (Keysizes free)
 - Named Elliptic Courves
 - * secp384r1
 - * sect409k1
 - * secp521r1
- Hashing
 - sha384
 - sha512
 - tiger192



6.1.7. Usability



6.2. Protocol Specification



6.2.1. MURBs



6.3. Protocol implementation



6.3.1. Header block



6.3.1.1. Ephemeral Identity



6.3.1.2. Requests



6.3.1.3. Replys



6.3.1.4. Crypto Puzzle



6.3.2. Main block



6.3.2.1. Routing Blocks



6.3.2.2. Routing Log Block



6.3.2.3. Reply Block



6.3.2.4. payload Block



6.4. Protocol Analysis



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 IV 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 offers according to the ASN.1 the possibility to store onionized information in an unencrypted form. This is meant for debuing purposes. At no point this should be used in a production environment.

The protocol described in the next chapter is independent from routing. At the moment capabilities include SMTP and XMPP. The protocol may be extended by adding new transport layer capabilities and their addressing schemes.

7. 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.

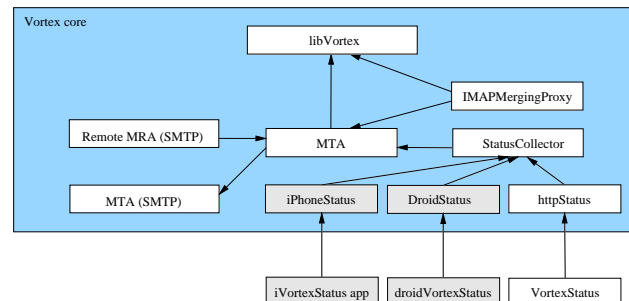


Figure 7.1.: Overview of the Vortex modules

7.1. Protocol Description

7.2. Accounting

7.3. Message Flows

7.4. Considerations for Building Messages

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.

7.4.1. Ephemeral identities



7.4.2. Timing of messages



7.4.3. Building Diagnostic Paths



7.4.3.1. Implicit Diagnostic



7.4.3.2. Automatic Explicit Diagnostic



7.4.3.3. On-Demand Explicit Diagnostic



7.5. Considerations for Routing Messages

7.5.1. 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.

7.6. 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 an affordable lightweight device which offers all required capabilities.

8. Security Analysis



9. Additional Considerations



9.1. Storage of Messages and queues

The storage of messages sent through 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 mail 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.



9.2. Economy of transfer



Part III.

Discussion



10. Anonymity



10.1. Effects of anonymous communication on behaviour



[27]

Part IV.

Appendix

ASN.1 representation of the protocol

Listing 1: ASN.1 representation of the protocol

```
1  -- FIXME ADAPT NeoScript
2
3  -- encryption: as specified in the key. If not specified default mode is ECB and default padding is PKCS1Padding
4
5  -- States: Tuple()=Value() [validity; allowed operations] {Store}
6  -- Tuple(identity)=Value(messageQuota,transferQuota,sequence of Routingblocks for Error Message Routing) [validity; Requested at creation; may be extended upon request] {identity}
7  -- Tuple(Identity,Serial)=maxReplays ["valid" from Identity Block; from First Identity Block; may only be reduced] {IdentityReplayStore}
8
9  Message-Blocks DEFINITIONS EXPLICIT TAGS ::=
10 BEGIN
11
12 -- define constants
13 maxSerial          INTEGER ::= 4294967295 -- maximum serial number
14 maxChunkSize       INTEGER ::= 4294967295 -- maximum size of a message chunk
15 maxNumberOfReplays INTEGER ::= 65535      -- maximum number of replays
16 maxNumberOfRequests INTEGER ::= 8         -- maximum number of administrative requests
17
18 Message ::= SEQUENCE {
19   header      IdentityBlock ,
20   blocks      CHOICE {
21     encrypted [1101] OCTET STRING, -- contains encrypted UnencryptedBlocks structure; Decryption key is in identity block [decryptionKey]
22     -- it is not allowed to use plain in production environments; This is for testing and analysis only
23     plain     [1102] UnencryptedBlocks -- should not be used except for internal diagnostic purposes
24   }
25 }
26
27 -- represents the identity of the rights owner
28 IdentityBlock ::= SEQUENCE {
29   headerKey [1000] OCTET STRING OPTIONAL, -- contains SymmetricKey encrypted with recipient nodes public key
30   identityBlock CHOICE {
31     encrypted [1001] OCTET STRING,
32     -- it is not allowed to use plain in production environments; This is for testing and analysis only
33     plain     [1002] IdentityPayloadBlock ,
34   },
35   identitySignature OCTET STRING -- contains signature of Identity [as stored in identityBlock; signed identityBlock without Tag]
36 }
37
38 IdentityPayloadBlock ::= SEQUENCE {
39   -- Public key of the identity representing this transmission
40   identityKey AsymmetricKey ,
41
42   -- serial identifying this block
43   serial      INTEGER (0..maxSerial),
44
45   -- number of times this block may be replayed (Tuple is identityKey,serial while
46   maxReplays  INTEGER (0..maxNumberOfReplays),
47
48   -- subsequent Blocks are not processed before valid time.
49   -- Host may reject too long retention. Recommended validity support >=1Mt.
50   valid       UsagePeriod ,
51
52   -- represents the chained secret which has to be found in subsequent blocks
53   -- prevents reassembly attack
54   forwardSecret [2000] ChainSecret OPTIONAL,
55
56   -- contains SymmetricKey encrypted with private key of identityKey
57   -- encryption is done as proof of identity (identity hijack protection)
58   decryptionKey OCTET STRING, -- contains (encrypted) DER encoded ASN1BitString with key representation
59
60   -- contains the MAC-Algorithm used for signing
61   hash          MacAlgorithm ,
62
63   -- contains administrative requests such as quota requests
64   requests      SEQUENCE (SIZE (0..maxNumberOfRequests)) OF HeaderRequest ,
65
66   -- padding and identifier required to solve the cryptopuzzle
67   identifier [2001] INTEGER (0..maxSerial) OPTIONAL,
68   padding    [2002] OCTET STRING OPTIONAL -- This is for solving crypto puzzles
69 }
70
71 UnencryptedBlocks ::= SEQUENCE {
72   -- contains routing information (next hop) for the payloads
73   -- FIXME how handle multiple payloads
74   routing [3000] RoutingBlock OPTIONAL,
75
76   -- contains encrypted log data of the data traveling
77   routingLog [3010] RoutingLogBlock OPTIONAL,
78
79   -- contains replays to header requests (eg. quota and identity handling)
80   reply [3020] ReplyBlock OPTIONAL,
81
82   -- contains the actual payload
83   payload [3100] SEQUENCE (SIZE (0..128)) OF PayloadChunk
84 }
85
86
87 -- represents the building and sending process for the next hop
88 RoutingBlock ::= SEQUENCE {
89
90   -- contains the next recipient in sequence
91   recipient NodeSpec,
92
93   -- contains the period when the payload should be processed
94   -- Router might refuse to long queue retention
95   -- Recommended support for retention >=1h
96   queueTime UsagePeriod ,
97
98   nextHop SEQUENCE (SIZE (0..128)) OF NextHopBlock, -- encrypted next RoutingBlocks for the payload
99
100   -- contains the secret of the identity block (if any)
101   forwardSecret [111] ChainSecret OPTIONAL,
102 }
```

ASN.1 representation of the protocol

```

103  -- contains a routing block which may be used when sending error messages back to the quota owner
104  -- this routing block may be cached for future use
105  replyBlock [131] RoutingBlock OPTIONAL,
106
107  -- This section is required if payload is routed with a prebuilt RB (
108  -- Messages MAY always request recompression (otherwise the message is identifiable from a non reply routing block)
109  decryptionKey [200] SEQUENCE (SIZE (1..2)) OF SymmetricKey OPTIONAL,
110  encryptionKey [201] SymmetricKey OPTIONAL,
111
112  -- contains information for building replays
113  cascade [300] SEQUENCE (SIZE (0..255)) OF CascadeBuildInformation
114 }
115
116 NextHopBlock ::= SEQUENCE {
117   nextIdentityBlock [13100] OCTET STRING,
118   nextRoutingBlock [13200] OCTET STRING,
119   nextReplyBlock [13300] OCTET STRING OPTIONAL,
120   nextErrorReplyBlock [13400] OCTET STRING OPTIONAL
121 }
122
123 PayloadOperation ::= SEQUENCE {
124   operation PayloadType,
125   thisid [12010] SEQUENCE (0..128) OF INTEGER (0..65535)
126 }
127
128 PayloadType ::= CHOICE {
129   randomPayload [100] RandomPayloadOperation,
130   splitPayload [150] SplitPayloadOperation,
131   mergePayload [200] MergePayloadOperation,
132   xorPayload [250] XorPayloadOperation,
133   encryptPayload [300] EncryptPayloadOperation,
134   ...
135 }
136
137 CascadeBuildInformation ::= SEQUENCE {
138   encryptionKey SymmetricKey,
139   secret ChainSecret,
140   payloadOp PayloadOperation,
141   ...
142 }
143
144 PercentSizeType ::= SEQUENCE {
145   fromPercent REAL (0..100),
146   toPercent REAL (0..100)
147 }
148
149 AbsoluteSizeType ::= SEQUENCE {
150   fromAbsolute INTEGER (0..maxChunkSize),
151   toAbsolute INTEGER (0..maxChunkSize)
152 }
153
154 SizeType ::= SEQUENCE {
155   reference INTEGER (0..65535),
156   size CHOICE {
157     percent [15001] PercentSizeType,
158     absolute [15101] AbsoluteSizeType
159   }
160 }
161
162 RandomPayloadOperation ::= SEQUENCE {
163   size SizeType
164 }
165
166 SplitPayloadOperation ::= SEQUENCE {
167   originalId INTEGER (0..65535),
168   firstSize SizeType,
169   secondId INTEGER (0..65535)
170 }
171
172 MergePayloadOperation ::= SEQUENCE {
173   -- FIXME
174 }
175
176 XorPayloadOperation ::= SEQUENCE {
177   -- FIXME
178 }
179
180 EncryptedRoutingLogBlock ::= OCTET STRING -- contains symmetrically encrypted RoutingLogBlock
181 RoutingLogBlock ::= SEQUENCE {
182   routingLog SEQUENCE (SIZE (0..16)) OF RoutingLog,
183   nestedRoutingInformationBlock EncryptedRoutingLogBlock
184 }
185
186 RoutingLog ::= SEQUENCE {
187   nodeIdIdentifier IA5String,
188   time GeneralizedTime,
189   code ErrorCode,
190   information IA5String
191 }
192
193 IdentityReplayStore ::= SEQUENCE {
194   replays SEQUENCE (SIZE (0..4294967295)) OF IdentityReplayBlock
195 }
196
197 IdentityReplayBlock ::= SEQUENCE {
198   identity AsymmetricKey,
199   valid UsagePeriod,
200   replaysRemaining INTEGER (0..4294967295)
201 }
202
203 IdentityStore ::= SEQUENCE {
204   identities SEQUENCE (SIZE (0..4294967295)) OF IdentityStoreBlock
205 }
206
207 IdentityStoreBlock ::= SEQUENCE {
208   valid UsagePeriod,
209   messageQuota INTEGER (0..4294967295),
210   transferQuota INTEGER (0..4294967295),
211   identity [1001] AsymmetricKey OPTIONAL, -- if omitted this is a node identity
212   nodeAddress [1002] NodeSpec OPTIONAL, -- if omitted own identity key
213   nodeKey [1003] SEQUENCE OF AsymmetricKey OPTIONAL, -- Contains the identity of the owning node; May be omitted if local node
214   routingBlocks [1004] SEQUENCE OF RoutingBlock OPTIONAL
215   ...
216 }
217
218 -- contains a node spec of a routing point
219 -- At the moment either smip:<email> or xmpp:<jabber>
220 NodeSpec ::= IA5String
221
222 ChainSecret ::= INTEGER (0..4294967295)
223

```

```

224 -- FIXME define requests
225 HeaderRequest ::= CHOICE {
226   identity      [0] HeaderRequestIdentity,
227   capabilities  [1] HeaderRequestCapability,
228   messageQuota [2] HeaderRequestIncreaseMessageQuota,
229   transferQuota [3] RequestIncreaseTransferQuota,
230   quotaQuery   [4] HeaderRequestQueryQuota,
231   ...
232 }
233
234 ReplyBlock ::= CHOICE {
235   identity      [0] ReplyIdentity,
236   capabilities  [1] ReplyCapability,
237   ...
238 }
239
240 HeaderRequestIdentity ::= SEQUENCE {
241   identity AsymmetricKey,
242   period UsagePeriod,
243   ...
244 }
245
246 ReplyIdentity ::= SEQUENCE {
247   challenge BIT STRING, -- bit sequence at beginning of hash from encrypted identity block
248   hash      MacAlgorithmIdentifier,
249   valid     UsagePeriod,
250   identifier INTEGER (0..4294967295),
251   ...
252 }
253
254 HeaderRequestQueryQuota ::= SEQUENCE {
255   identity AsymmetricKey,
256   ...
257 }
258
259 HeaderRequestIncreaseMessageQuota ::= SEQUENCE {
260   identity AsymmetricKey,
261   messages INTEGER (0..4294967295),
262   ...
263 }
264
265 RequestIncreaseTransferQuota ::= SEQUENCE {
266   identity AsymmetricKey,
267   size     INTEGER (0..4294967295),
268   ...
269 }
270
271 HeaderRequestCapability ::= SEQUENCE {
272   period UsagePeriod,
273   ...
274 }
275
276 ReplyCapability ::= SEQUENCE {
277   cypher SEQUENCE (SIZE (2..256)) OF CypherSpec,
278   maxTransferQuota INTEGER (0..4294967295),
279   maxMessageQuota  INTEGER (0..4294967295),
280   supportedProtocol SEQUENCE OF Protocol,
281   ...
282 }
283
284 CypherSpec ::= SEQUENCE {
285   asymmetric AsymmetricAlgorithmIdentifier,
286   symmetric   SymmetricAlgorithmIdentifier,
287   mac         MacAlgorithmIdentifier
288 }
289
290 Protocol ::= ENUMERATED {
291   smtp (100),
292   xmmp (110),
293   ...
294 }
295
296 ErrorCode ::= ENUMERATED {
297   -- System messages
298   ok (2001),
299   transferQuotaStatus (2101),
300   messageQuotaStatus (2102),
301   -- protocol usage failures
302   transferQuotaExceeded (3001),
303   messageQuotaExceeded (3002),
304   identityUnknown (3101),
305   messageChunkMissing (3201),
306   messageLifeExpired (3202),
307   -- Mayor host specific errors
308   hostError (5001),
309   ...
310 }
311
312 PayloadChunk ::= SEQUENCE {
313   offset INTEGER (0..MAX),
314   routingBlockIdentifier [0] SEQUENCE (SIZE (0..255)) OF INTEGER, -- FIXME limit integer range
315   payload [100] OCTET STRING
316 }
317
318 -- Compatible to PrivateKeyUsagePeriod taken from RFC3280
319 UsagePeriod ::= SEQUENCE {
320   notBefore [0] GeneralizedTime OPTIONAL,
321   notAfter  [1] GeneralizedTime OPTIONAL
322 }
323
324 -- adapted from RFC3280
325 SymmetricAlgorithmIdentifier ::= SEQUENCE {
326   algorithm SymmetricAlgorithm,
327   parameter AlgorithmParameters OPTIONAL
328 }
329
330 AsymmetricAlgorithmIdentifier ::= SEQUENCE {
331   algorithm AsymmetricAlgorithm,
332   parameter AlgorithmParameters OPTIONAL
333 }
334
335 MacAlgorithmIdentifier ::= SEQUENCE {
336   algorithm MacAlgorithm,
337   parameter AlgorithmParameters
338 }
339
340 SymmetricAlgorithm ::= ENUMERATED {

```

```

345     aes128      (1000),
346     aes192      (1001), — optional support
347     aes256      (1002),
348     camellia128 (1100),
349     camellia192 (1101), — optional support
350     camellia256 (1102)
351 }
352
353 AsymmetricAlgorithm ::= ENUMERATED {
354     rsa      (2000),
355     dsa      (2100),
356     secp384r1 (2500),
357     sect409k1 (2501),
358     secp521r1 (2502)
359 }
360
361 MacAlgorithm ::= ENUMERATED {
362     sha384 (3000),
363     sha512 (3001),
364     — FIXME check AEAD
365     tiger192 (3100)
366 }
367
368 ECCurveType ::= ENUMERATED{
369     secp192r1,
370     sect163k1,
371     sect163r2,
372     secp224r1,
373     sect233k1,
374     sect233r1,
375     secp256r1,
376     sect283k1,
377     sect283r1,
378     secp384r1,
379     sect409k1,
380     sect409r1,
381     secp521r1,
382     sect571k1,
383     sect571r1,
384     ...
385 }
386
387 AlgorithmParameters ::= SEQUENCE {
388     keySize [10000] INTEGER (0..65535) OPTIONAL,
389     curveType [10001] ECCurveType OPTIONAL,
390     ...
391 }
392
393 — Symmetric key
394 SymmetricKey ::= SEQUENCE {
395     keyType SymmetricAlgorithmIdentifier,
396     key OCTET STRING
397 }
398
399 — Asymmetric Key
400 AsymmetricKey ::= SEQUENCE {
401     keyType AsymmetricAlgorithmIdentifier,
402     publicKey [1] OCTET STRING,
403     privateKey [2] OCTET STRING OPTIONAL
404 }
405
406 pkcs-1 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) 1 }
407
408 END

```

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 [9]. 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 (Iol) FIXME

LMTP 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 forwards 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 forwards 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.

"[37, 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 [19].

Storage A store to keep data. It is assumed to be

Glossary

temporary or persistent in its nature.

user FIXME

UBE FIXME

Bibliography

- [1] 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. URL: <http://www.abortz.com/papers/k-anon.pdf> (cit. on p. 17).
- [2] *AMQP v1.0*. AMQP.org, 2011. URL: <http://www.amqp.org/confluence/display/AMQP/AMQP+Specification> (cit. on p. 13).
- [3] Banks Andrew and Gupta Rahul. *MQTT*. en. OASIS, 2014. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf> (cit. on p. 13).
- [4] Mike Belshe, Roberto Peon, and Martin Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. May 2015. DOI: 10.17487/rfc7540. URL: <https://rfc-editor.org/rfc/rfc7540.txt> (cit. on p. 13).
- [5] 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. 14).
- [6] *Campaign Monitor*. 2012. URL: <http://www.campaignmonitor.com/resources/will-it-work/email-clients/> (cit. on p. 25).
- [7] 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. 18, 20, 22).
- [8] 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. 21).
- [9] M. Crispin. *RFC3501 INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. IETF, 2003. URL: <http://tools.ietf.org/pdf/rfc3501.pdf> (cit. on p. 53).
- [10] M. Elkins. *RFC2015 MIME Security with Pretty Good Privacy (PGP)*. IETF, 1996. URL: <http://tools.ietf.org/pdf/rfc2015.pdf> (cit. on p. 1).
- [11] *Email Client Market Share*. 2014. URL: <http://emailclientmarketshare.com/> (cit. on p. 25).
- [12] *Fact Sheet Suite B Cryptography*. 2008. URL: http://www.nsa.gov/ia/industry/crypto_suite_b.cfm (cit. on p. 20).
- [13] 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, 14).
- [14] 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. 14).
- [15] R. Gellens and J. Klensin. *RFC4409 Message Submission for Mail*. IETF, 2006. URL: <http://tools.ietf.org/pdf/rfc4409.pdf> (cit. on p. 24).
- [16] 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. 21).
- [17] 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. 22).
- [18] 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 p. 19).
- [19] J. Klensin. *RFC5321 Simple Mail Transfer Protocol*. IETF, 2008. URL: <http://tools.ietf.org/pdf/rfc5321.pdf> (cit. on pp. 14, 23, 53).
- [20] 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. 20).
- [21] 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. 20).
- [22] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramanian. “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. 17).
- [23] 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. 20).
- [24] Jeffrey 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. 13).

- [25] Suresh Venkatasubramanian Ninghui Li Tiancheng Li. "t-Closeness: Privacy Beyond k-Anonymity and". In: (). URL: http://www.cs.purdue.edu/homes/li83/papers/icde_closeness.pdf (cit. on p. 17).
- [26] 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. 17, 18).
- [27] Tom Postmes, Russell Spears, Khaled Sakhel, and Daphne De Groot. "Social influence in computer-mediated communication: The effects of anonymity on group behavior". In: *Personality and Social Psychology Bulletin* 27.10 (2001), pp. 1243–1254. DOI: 10.1177/01461672012710001. URL: <http://psp.sagepub.com/content/27/10/1243.short> (cit. on p. 45).
- [28] 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. 25).
- [29] 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. 25).
- [30] 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. 21).
- [31] P. Resnick. *RFC5322 Internet Message Format*. IETF, 2008. URL: <http://tools.ietf.org/pdf/rfc5322.pdf> (cit. on p. 14).
- [32] 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. 19).
- [33] 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. 14).
- [34] 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. 14).
- [35] P. Saint-Andre. *RFC6120: Extensible Messaging and Presence Protocol (XMPP): Core*. IETF, 2011. URL: <http://tools.ietf.org/pdf/rfc6120.pdf> (cit. on p. 14).
- [36] 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> (cit. on p. 20).
- [37] A. Stevenson. *Oxford Dictionary of English*. Oxford reference online premium. OUP Oxford, 2010. ISBN: 9780199571123. URL: <http://www.oed.com> (cit. on p. 53).
- [38] Almon Brown Strowger. "Automatic Telephone-Exchange". en. Pat. 447918. Mar. 1891 (cit. on p. 1).
- [39] 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. 14).
- [40] UNHR. *International Covenant on Civil and Political Rights*. 1966. URL: <http://www.ohchr.org/en/professionalinterest/pages/ccpr.aspx> (cit. on p. 9).
- [41] Michael Waidner and Birgit Pfitzmann. "The dining cryptographers in the disco: Unconditional Sender and Recipient Untraceability". In: *Proceedings of EU-ROCRYPT 1989*. Springer-Verlag, LNCS 434, 1990. URL: http://www.semper.org/sirene/publ/WaPfl_89DiscoEngl.ps.gz (cit. on p. 21).

Index

Item of Interest, 10

Mail transport, *see* Message Transport