Sending unobservable messages across the internet

Martin Gwerder

Abstract—In this paper we introduce an unobservable message annonymisation protocol, named MessageVortex. It is based on the zero trust principle and a distributed peer-to-peer (P2P) architecture and avoids central aspects such as fixed infrastructures within a global network. It scores over existing work by blending its traffic into suitable existing transport protocols, thus making it next to impossible to block it without significantly affecting regular users of the transport medium. It furthermore requires no protocol-specific infrastructure in public networks and allows a sender to control all aspects of a message such as degree of anonymity, timing, and redundancy of the message transport without disclosing any of these details to the routing or transporting nodes.

Index Terms—Data privacy, Message systems, Anonymity, Security

1 Introduction

Since whistle blower Edward Snowden disclosed documents, it seems generally accepted that global monitoring of Internet traffic is conducted. According to these documents (verified by NRC) NSA infiltrated more than 50k computers with malware to collect classified or personal information. They furthermore infiltrated telecom operators such as Belgacom to collect data, and targeted high members of governments even in associated states.

A message sent throughout the Internet must, even when perfectly encrypted, disclose at least the recipient to the router transporting a message. The sender can be identified by the return path or is identifiable by following the source of packets. Meta information is valuable, because frequency and message size disclose important facts about the association and intensity of the relationship of involved parties.

This paper addresses the above-mentioned problems of traffic monitoring by introducing a new protocol called MessageVortex. Within MessageVortex we consider the whole network as untrusted with the exception of the sending and receiving node. MessageVortex does not leak routing information as only the immediate peers are known to a node. The protocol is able to sustain anonymity [**?**] even under harsh assumptions such as an adversary possessing a huge but limited funding, unlimited monitoring capability on the network and a considerable number of own nodes. For a more precise adversary model, see appendix C.

Numerous attempts such as in [1], [2], [3], [4], [5], [6] have been made to anonymise message flow. But most of them have problems as they rely at least on the partial trust in the nodes routing the messages, or some central infrastructures [7], [8], [9], [10]. Exit and entry points are important as they may leak information which is otherwise well hidden within the network. By degrading the network, message flows can be redirected and information extracted from the new flows. Additionally, a dedicated transport protocol is easy to block since their implementation can be easily identified by used ports or some protocol properties. Furthermore, most approaches require infrastructure with fixed addressing in the internet, rendering owners vulnerable.

All papers analysed for this work introduced a new transport layer solving these problems. Only TOR defined an additional transporting mechanism which may be used as an alternate transport medium between two defined nodes to avoid detection. In our approach we decouple the routing layer from the transport layer completely. By doing so we introduce new degrees of complexity to attack scenarios, as messages may use any common transport protocol of the used network.

Our work consists of a routing layer which is completely P2P based without any central protocol specific infrastructure. Any node is a routing node and may be an endpoint. There is no implicit or explicit trust in any particular system of the network. Decoy traffic generation is controlled by the original sender of a message. Even a decoy traffic generating node is unable to differentiate between message and decoy traffic as a Solomon-Reed algorithm is used to blow the message up by adding redundancy information. This redundancy information may be decoy traffic or later required to rebuild data blocks. The redundant blocks are always encrypted and a multitude of the cyphers block size as they are padded before splitting. This eliminates the need of padded encryption at block level. This fact makes it very hard to apply brute force in order to decrypt the content. This is due to the fact that padding no longer hints gives whether decryption has been successful or not.

As transport media we use common, well known store-and-forward-based protocols. By doing so the routing logic has no affiliation to the transport layer. Literally any free-mailer email address or chat account may be converted into a transport media for our protocol without any modification required on the server side. This makes the network very agile on one side at the cost of reliability, as nodes may suddenly appear or disappear. To counter this phenomenon, we are able to introduce a high degree of redundancy if required and wished by the routing block builder.

Using the MessageVortex protocol, any device with a latent or permanent connectivity to the Internet may act as routing node.

By applying the zero trust model we give full control of all traffic to the original sender of the message. He controls message flow, redundancy, degree of anonymity, timing, and many more aspects of the message transport throughout the whole network. This is done without disclosing any of these parameters to the participating nodes as they are encoded in the operations and only visible to the node executing them. The operations itself are chosen in such a way that they do not reveal the nature of the traffic.

To limit possibilities of denial-of-service (DoS) within the system and guarantee an efficient handling of messages, MessageVortex nodes (in short "node") rely on unlinked, ephemeral identities which are created in a proof of work system (PoW). While it is technically easy to use a node, it is hard to carry out traditional attacks against them as all transactions have to be pre-authenticated with PoW puzzles. The amount of work required to disrupt services or conduct traditional attacks

against the system grows significantly due to the non linear growth of calculation power required when maintaining more ephemeral identities. It is, however, still possible to exhaust external resources such as network bandwidth.

1.1 Previous Work

Generally not many technologies are usable to achieve anonymity or unlinkability as defined in [**?**]. Most analysed protocols use relays [11], mixes [11], or Dining-Cryptographers-related-networks [12] or their variants to achieve anonymisation. Numerous protocols have evolved from these technologies:

- *TOR* [4]: Mixer-based infrastructure for tunnelling TCP-based protocol streams. TOR is a synchronous or near synchronous routing system. The anonymisation is based on mixing by using a statical path consisting of an entry node, an exit node and at least three more intermediate nodes.
- *mixmaster* [3]: A type-II remailer where all mixes may choose the path on their own logic.
- *Babel* [2]: Mixer-based remailer where the sender chooses the path and sends an onionised message.
- *Mixminion* [1]: A type-III remailer offering sender anonymity. Unlike their predecessors, it is no longer based on the SMTP transport protocol. This system requires at least a centralised directory infrastructure.
- *Freehaven* [5]: A distributed storage system. The system offers anonymous document storage. To receive a document, a hash of a public key used to sign the document must be known. Known documents may be identified and owners of infrastructure might be held responsible if hosting such well known, forbidden documents.
- *Freenet* [13]: Freenet is an anonymous, distributed data storage system. The system does not trust any server. Instead a reputation system is used. This system has attracted very little attention from the researcher community.
- *Herbivore* [6]: A DC-net-based protocol without client implementation.
- $\mathcal{P}^5$ [14]: There is a simulator available for this protocol. Real world implementations do not exist and therefore no attack schemes have been elaborated so far.
- I^2P(geti2p.net): P2P-based pseudonymous protocol allowing TCP and UDP streams to be tunnelled synchronously or near-synchronously. Unlike TOR, I^2P works pseudonymously and mixes using packet switching.

Our protocol differs from these works in several ways. There is no central network infrastructure. There are no entry or exit nodes which might be blocked. All nodes including the sender and the recipient are treated equal. The number of nodes, the traffic to be generated, anonymity sets, timing of the message, redundancy in message transmission, and size of all packages to be sent along is solely decided by the builder of a routing block. The builder is normally synonymous to the sender but might be the recipient of a message in case of a reply block. Furthermore, there is no dedicated transport protocol. Instead, MessageVortex messages (in short "vmessages") are embedded in other existing Internet protocols. The traffic itself is mixed by operations. As traffic is generated reproduceable, either by adding redundancy information or by using PRNG with a defined seeding, decoy traffic cannot be differentiated from required blocks.

2 Methods and Material

We define the protocol on three different layers:

- *Blending layer:* in this layer we embed MessageVortex message into the transport protocol.
- *Routing layer:* this layer applies the logic to the message routing and prepares the message for the blending layer.
- *Accounting layer:* this layer is a DoS and misuse protection. It keeps track of the transfers for each ephemeral identity and makes sure that queue and storage capacity are efficiently handled.

These three layers are connected through a fourth existing layer. This layer is based on one or more store-and-forward based, common internet transport protocols. Protocols on this layer we refer in general as transport protocols. It is important to note that no modification is applied to the transport protocol to accommodate vmessages.

All cryptographic operations such as encryption, decryption, hashing, or random number generation do not rely on a single algorithm. The protocol is able to signal what capabilities a node has and how exactly a message should be processed. This makes the protocol very robust if a used algorithm is broken. For this reason, we defined for each capability at least two algorithms which depend on different mathematical puzzles (e.g. "integer factorisation problem" versus "discrete logarithms problem"). This introduces a redundancy in algorithms, allowing a sender to switch between algorithms if required.

2.1 Protocol Layers

2.1.1 Transport

The transport layer provides the Internet infrastructure. Unlike in most other approaches such as [4], [13], [14] this layer is not protocol specific. We use already existing, symmetrically built store and forward protocols. Attributes such as anonymity do not rely on the security of this layer.

By using this approach we remove the need for shaky technologies such as TCP or UDP hole punching to connect peer partners. It furthermore makes the use of "mostly connected" clients such as mobile phones or DSL connections suitable for this protocol. This is because our transport endpoints are always reachable within the global network. The routing nodes may disconnect from time to time without affecting reachability of an mix.

Protocols on this layer are typically well known and frequently used. They have no prerequisite for encryption or privacy and are store-and-forward based protocols with routing capabilities.

2.1.2 Blending

This layer is a translation-only layer and embeds vmessages from the routing layer in transport protocol messages. Incoming vmessages are extracted from the transport layer and passed to the routing layer. Messages can be identified by picking a potential vmessage block up and start decyphering k_{p_N} using its private key $k_{h_N}^{-1}$. If decryption succeeds a vmessage block is found. This makes it impossible for an adversary to detect the presence of a vmessage without the hosts private key $k_{h_N}^{-1}$.

Protocol features such as anonymity or redundancy do not rely on this level. This layer embeds messages within the transport layer in such a way that an adversary is no longer able to identify vmessages from regular transport layer messages. Good blending is achieved if transport layer censorship measurements such as application level firewalls are unable to detect the difference between real world messages and vmessages. In an ideal application, this applies to censorship applied by humans as well as censorship applied on the base of algorithms.

In a real scenario, it is hard to achieve human proof censorship circumvention. If not done with care, problems as described in [15] arise. It is in our case not necessary as human censorship

is very costly and slow compared to algorithms. Human censorship is too slow for real time censorship. Our transport layer is by definition frequently used for regular communication. We always consider an algorithm-based censorship as existing.

Currently, the specification of this layer is limited to the two capabilities "embed with offset" and "F5".

"Embed with offset" is a plain embedding of a block in a file attached to a message. The offset allows issuing first a valid header of some sort in order to improve blending (e.g. for a PCM-encoded WAV file). While this is considered a very weak protection, analysis to detect such a file on a global transport scale is very demanding due to the sheer mass to be analysed.

"F5" means applying the F5 algorithm to hide a message within a random suitable jpeg image. "F5" is one of the very few steganographic works which have a real world implementation and attracted at least some interest in the research community. In [16], an approach to detect embedded information in steganographically modified images is presented. To obtain this information, a considerable effort in terms of calculation power is required. This makes it impractical for real time censorship on our scale. It does however allow messages to be identified. Furthermore, It only discloses the fact that F5 is being used. It does not leak the content of a message, its immediate sender (apart from a socket), or a message size.

2.1.3 Routing

The routing layer is the mixer of the system. It processes messages extracted by the blending layer and is supported by the accounting layer. Any related set of messages is processed by the routing layer by recombining payload with operations defined in section 2.5. Due to the nature of these operations, a node is unable to tell whether the traffic flow processed is decoy traffic or an actual part of the message flow.

If a message is processed a new vmessage is generated and passed on to the blending layer.

For a more precise working of the routing process see section 2.4.

2.1.4 Accounting

The accounting layer protects a node from being overloaded or misused. Every sender must first apply for an ephemeral identity which is limited in lifetime. This is done by a proof of work algorithm. When an ephemeral identity is created the owner of the identity may route vmessages through a node. The ephemeral identity is assigned with message and size transfer quotas. Any identity may apply for a raise of quota as long as it is not expired. It is up to the node to decide whether a raise of quota is acceptable or not. If rejected, any sender might try to apply for a new ephemeral identity.

Due to the costs of maintaining multiple identities and their parental identities for anonymity of the original sender, the number of identities grows exponentially when growing a network of ephemeral identities. A sender might either introduce a new node to cut identity costs or maintain at higher identity costs a single node.

2.2 Protocol Outline

We define a protocol block which has an inner block structure as shown in Fig 1.

These blocks are passed from node to node. All blocks are binary proof, which means that the same block sent twice will always result in exactly the same bit layout. There is no room for a misbehaving node to tag the block within it without compromising the message. The message as a whole is replay protected. Routing and header blocks are linked with a chain secret to avoid hijacking of header or routing blocks.

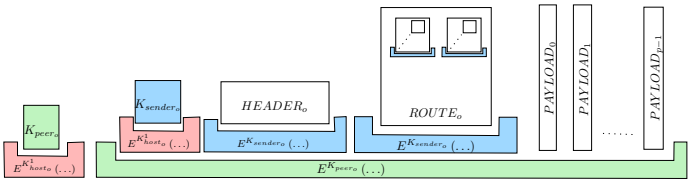

Fig. 1. Protocol block outline.

2.2.1 Message Keys

Every protocol block is protected by two symmetric keys key_{peer_N} (in short k_{p_N}) , key_{sender_N} (in short k_{s_N}) and the private part of an asymmetric host key $k^{-1}_{host_N}$ (in short $k^{-1}_{h_N}$). The public host key $k^1_{h_N}$ and both symmetric keys are known to the builder of the routing block structure.

This building is done by the sender. If using SURBs (Single Use Reply Blocks) or MURBs (Multi Use Reply Blocks) it is done by the builder of the reply block.

The header is protected by the symmetric key k_{s_N} and is found in a preamble to the header protected by the receiving peer's private key $k^{-1}_{h_N}$. The key k_{s_N} is known to the routing block builder and the receiving node only. The receiving node obtains all important information protected by this key. k_{p_N} is known to two immediate peers and the builder of the routing block. The sending peer obtains k_{p_N} from the routing block, whereas the receiving peer acquires it in the $headerBlock$.

2.3 Pad Block

The pad block is a short block of a few bytes of padding content (first bytes of the first message block; fixed size per message; null padded) guaranteeing that all messages sent with the same routing block look different on the transport layer.

2.3.1 Header Block

The header block contains vital static information for the message disclosed to only one peer of the network. It is protected by key k_{s_N}. The minimally contained information can be described by the list $headerBlock_i := \langle sendingIdentity, serial_i, replayAttributes_i, key_{p_i}, chainSecret, signature, optionalOperations \rangle$.

2.3.2 Routing Block

A routing block can be expressed with the following recursive definition $routingBlock_i := \langle nexthopAddress, chainSecret, timingAttributes, E^{k_{s_{i+1}}}(headerBlock_{i+1}), E^{k_{s_{i+1}}}(routingBlock_{i+1}), payloadBuildInstructions_i, payloadId, optionalReplyBlocks \rangle$

2.3.3 Payload Block

A payload block is any number of bytes representing parts of a message, decoy traffic or a control block.

2.4 Message Processing

Unlike with a traditional mix system, a node has no choice of sending. It purely relies on the message processing facilities. A message is either handed over to the transport layer by the blending layer or may be induced internally (if the local node is the sender).

First, the preamble to the header is extracted. This proves that the sender possesses the public key of the node and contains the sender key k_{s_i}. With this information, the node opens the $headerBlock$ revealing information regarding the ephemeral identity of the original sender. Based on the information given in the relatively small header, the transport layer may decide whether further processing is desired or not. If desired, the node extracts the key k_{p_i} and decrypts the rest of the message, which is considerably larger containing routing and payload information.

The routing block may contain instructions on processing information contained in this or any message related to this message and identity. These instructions are encoded in so-called "Operations" as specified in section 2.5 and may be any combination of them. As soon as the time arises for a routing block to be processed, the operations building the new message blocks are executed. If all prerequisites are satisfied, the new payload blocks are built, concatenated with the new routing block, pad, and header block. This resulting block is encrypted with $k_{p_{i+1}}$ and prepended with the preamble. The built block is passed to the blending layer with the blending specification and the target address.

It is important to note that the blending specification contains vital information about how the message must be blended but not how the carrier message looks like. By doing so we avoid abuse of the blending layer (eg. sending plain text spam through the MessageVortex system).

2.5 Operations

The operations are designed in such a way that they allow variance of message size without telling anyone, including the generator, which message part is used later. They include features to protect message content from bugging.

Some of the operations require a pseudo random number generator (PRNG). This PRNG is defined in appendix B. The definition of a reproduceable PRNG to be used by messages is important as we have to achieve binary proof messages.

All interactions are non-interactive. Interactive operations such as DC-nets do add more complexity to the system. Behavioural analysis can be used to identify interactive operations.

This is the reason why DC-nets are not used. Theoretically, it is possible reflect them as a single operation by calculating the answer and then broadcasting the answer. In practice, this fails due to the non-existence of efficient, reliable multicast networks. There are however attempts to apply DC-nets to real protocols [17].

2.5.1 addRedundancy *and* removeRedundancy *Operation*

This operation is based on a modified Reed-Solomon redundancy function in order to accommodate the anonymity needs of this function. The Reed-Solomon function as defined in appendix A offers a varying number of redundant checksum blocks. When sending these blocks into multiple directions no mixing node is able to tell where the original message is being rebuilt. The general inner workings are described in Fig 2.

We define a function $\mathrm{addRedundancy}_{n,m}(\mathbf{M}, k_1 \dots k_m)$ where M denotes the message, n the number of total output blocks, m the number of redundancy blocks wheras $m < n$, k the encryption key and scheme to be used, and bs_k the block size required to accommodate scheme and key size described by k. It is important to note that the number of true data blocks is $d = n - m$, while the rest of the output blocks are redundant information.

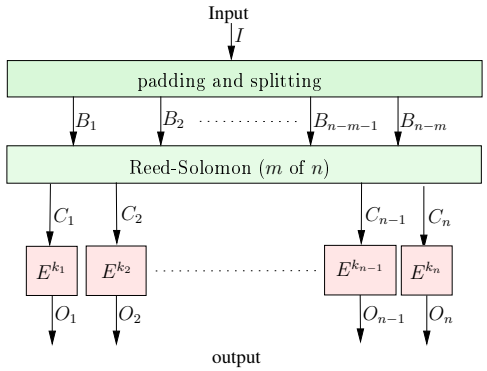

Fig. 2. AddRedundancy Operation

By encrypting all output blocks individually we make sure that no node having access to enough blocks may rebuild the data stream without the routing block builder's consent.

The message is length prefixed with a big endian 64 bit unsigned integer number and padded in such a way that $8 + \mathrm{len}(M) + \mathrm{len}(padding) \bmod bs_k = 0$. As padding stream, we take the output of $\mathrm{prng}_i\left(\left\lceil \frac{8+\mathrm{len}(M)}{b_s} \right\rceil b_s n\right)$. The first 64 bytes of the message (padded with 0 if required) are taken as initialiser i for the PRNG function. By preparing our message block in such a way, we guarantee that the output blocks are encryptable without further padding and that the output of all $addRedundancy$ functions is binary proof. If stream cyphers are used as output cyphers, then padding is not required.

The reverse function for addRedundancy is called $\mathrm{removeRedundancy}_{m,n}(B_1 \dots B_m, k_1 \dots k_m) = \mathbf{M}$ and recovers the original data stream if enough blocks (at least m) and respective valid keys are provided.

2.5.2 splitPayload *and* mergePayload *Operation*

The splitPayload and mergePayload operations split and merge payload blocks (pb_N)into two chunks of different or equal sizes respectively joins them. We define the functions as follows:

If $\mathrm{len}(pb_1)$ expresses the size of a payload block called pb_0 in bytes then the two resulting blocks of the $splitPayload$ Operation pb_1 and pb_2 have to follow the following rules:

$$\mathrm{splitPayload}(f, pb_0) = \langle pb_1, pb_2 \rangle \quad (1)$$
$$\mathrm{startsWith}(pb_0, pb_1) \quad (2)$$
$$\mathrm{endsWith}(pb_0, pb_2) \quad (3)$$
$$\mathrm{len}(pb_2) = \lfloor \mathrm{len}(pb_0) \cdot f \rfloor \quad (4)$$
$$\mathrm{len}(pb_0) = \mathrm{len}(pb_1) + \mathrm{len}(pb_2) \quad (5)$$

respectively

$$\mathrm{mergePayload}(pb_1, pb_2) = pb_0 \quad (6)$$
$$\mathrm{startsWith}(pb_0, pb_1) \quad (7)$$
$$\mathrm{endsWith}(pb_0, pb_2) \quad (8)$$
$$\mathrm{len}(pb_0) = \mathrm{len}(pb_1) + \mathrm{len}(pb_2) \quad (9)$$

2.5.3 xorSplit *and* xorMerge *Operation*

xorSplit and xorMerge are low cost obfuscation operations. These operations may be applied if a block is passed on without any required operation or as one-to-two blocks redundancy generating function.

The operations are defined as follows:

$$\begin{aligned} \text{xorSplit}(pb_0) &= \langle pb_1, \text{prng}_i(\text{len}(pb_0))\rangle && (10)\\ pb_1 &= pb_0 \oplus \text{prng}_i(\text{len}(pb_0)) && (11)\\ \text{xorMerge}(pb_1, pb_2) &= \langle pb_0\rangle && (12)\\ pb_0 &= pb_1 \oplus pb_2 && (13) \end{aligned}$$

2.5.4 encrypt *and* decrypt *Operation*

encrypt and decrypt are used as message obfuscation operations. These operations may be applied if a block is passed on without any required operation. They minimise the risk for a known plain text attack to a MessageVortex block. Both operations are defined as a padded or unpadded symmetrical encryption. *spec* is the encryption specification and key provided by the routing block.

The operations are defined as follows:

$$\begin{aligned} \text{encrypt}(pb_0) &= pb_1 && (14)\\ \text{len}(pb_1) &\geq \text{len}(pb_0) && (15)\\ \text{decrypt}(pb_1) &= pb_0 && (16) \end{aligned}$$

2.6 Protocol Bootstrapping

In order to allow bootstrapping of the protocol, any node may reveal a very small, fixed number of nodes known to it. This allows fresh nodes to bootstrap their knowledge about an existing network, given they know at least one node willing to reveal other nodes.

Allowing this kind of bootstrapping has certain downsides. As no trust is given into the requester's identity, we have to be very careful here not to reveal the full network to any adversary. By applying the PoW and requiring an ephemeral identity, we assign a high cost to this operation.

Another downside is that it takes a long time for such a network to balance its loads if a network increases over time in size. Mature nodes concentrate more traffic on them than younger ones. This does however distribute more evenly over time if algorithms as shown in [18] are applied.

3 Results

Our protocol can be seen as tool set for creating and sending anonymised messages. The degree of anonymity and redundancy is created when building the routing block. This is why constraints on message building are important.

3.1 Message Building

Using previously defined operations we may build a message path. This path is typically built by first assigning an identity set I_k where k denotes the target identity. I_k is a static set of n ephemeral identities $I_k\langle eI_1 \dots eI_n\rangle$ which are always used to communicate with k. This set may be enriched with further m ephemeral identities when sending. An identity set is replaced with a different one as soon as ephemeral identities expire. Therefore, we apply a new anonymity set unrelated to the old one with each new set of ephemeral identities.

A full message graph including all traffic may have any type of complexity. Fig 3 shows a graph with a k-Anonymity of $k = 7$ of a message. It features 5 partially independent routes from source (edge 0) to the target (edge 1). One is highlighted in green. All involved nodes receive enough information to build the entire message if provided with the correct decoding instruction. X-axis shows involved nodes and y-axis denotes the sequence in time. In its background a graph with $k = 30$ and 20 partially independent paths is shown.

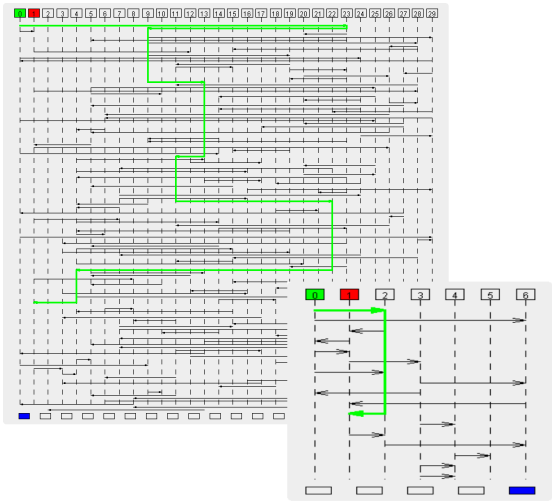

Fig. 3. Message graphs with different numbers of nodes

When building the message it has to be ensured that all nodes in I_k obtain enough information to rebuild the message. If an adversary is able to identify the full message flow and knows all the operations applied to the message except for those on the entry and exit node and at least a subset of $k = |I_{k_{uncompromised}}|$ where $k > 1$ exists then we are still at k-Anonymity as an absolute worst case scenario. Thus we can prove that attacks as described in [19] are of very limited use.

3.2 Attacking the Message Flow

In our thesis [18] we analyse various kinds of attacks. Such as illicit behaving nodes, hijacking of header and routing blocks, analysis on payload blocks, traffic replay, analysis on infrastructure, and analysis on operations. Results have shown that the protocol is very resistant against most kinds of attacks.

For block hijacking of a single block we can proof that probability for success is at least below $10E-11$. We can furthermore prove the effectiveness of replay protection even when assuming misbehaving nodes. Hijacking a routing block has very limited use however. Prepending an own identity block to a hijacked routing block would break the message path. Thus, allowing an adversary to reveal all blocks built from this routing block out of the message travelling through the adversary node. He does not gain any information about other blocks. He may extract how much traffic is generated on the manipulated node out of these blocks. The real amount of traffic is, however, higher as some operations may have failed due to missing blocks.

We can easily show the effectiveness of the tagging and bugging protection. A misbehaving node has no room to tag a message without compromising the message's integrity. A tagged message will be discarded at the first non-misbehaving node.

3.3 Routing Diagnosis

If an interruption of path is suspected, parts of the message may be obtained by the message block builder at any time. He may do this by either introducing fixed diagnostic paths into a routing block, which we refer to as implicit diagnosis, or he may send a second message picking up a block of the message at a node to be tested. This we refer to as explicit diagnostic.

Explicit diagnostics may be used as a kind of "receipt" from any node including but not limited to the terminal receiver of a message. Any block at any time of routing may be returned directly or indirectly to the original sender. Arrival of such a packet and content tells the sender at which point a message failed. If a diagnostic packet does not arrive, the routing block builder may build a diagnostic message picking up random packets on any suspected failing node.

4 Discussion

4.1 Comparison to Existing Systems

The following section gives a short comparison to existing systems. It shows that the solution defined in this paper covers a different approach and what problems are solved. It is important to note that this is not a ranking. It just outlines the differences between the system and shows where our system is different compared to existing solutions.

4.1.1 TOR

TOR is criticised for several things. Firstly, it is easy attackable if encryption is not used in the transported protocol. It relies on the trust in a centralized directory infrastructure. It is susceptible if more than $\approx 30\%$ of the nodes are controlled by an adversary as shown in [20]. Furthermore, timing analysis on entry and exit nodes are particularly easy due to the fact that TOR is a low latency network [21], [22]. Harvesting of nodes is possible (e.g. https://torstatus.blutmagie.de). Tor nodes are easily identifiable by traffic as shown in [23]. To avoid this detection TOR uses "pluggable transports". Unlike in MessageVortex, this is not used in general but only when specifically set up between two nodes.

MessageVortex tries to address these problems in multiple ways. First, there is no central infrastructure which defies the trust problem. There are no entry or exit nodes as all participating members are routers at the same time. Therefore, all problems related to entry and exit nodes do not exist. There is no dedicated transport protocol making the presence of vmessages hard to detect.

MessageVortex has several downsides compared to TOR. It is not suitable for real time communication due to its asynchronous operation. It is furthermore a closed system and only participating members may use it.

4.1.2 $\mathcal{P}^5$

In-depth analysis of $\mathcal{P}^5$ is very limited, as there is no true protocol specification but only a rough outline available. This outline specifies the messaging and the crypto operations only. It claims to be peer to peer, which would result in some kind of NAT (Network Address Translation) circumvention technology. This technology usually relies, at least partially, on a central infrastructure (e.g. for hole punching).

In contrast MessageVortex protocol is peer to peer but the transport layer is not. It misuses already existing infrastructure for transport. This makes it not susceptible to approaches against infrastructure unless our messages are identified and filtered. This may be corrected by applying different blending schemes for the transport layer. It furthermore removes the need for NAT hole punching and similar technologies.

4.1.3 I^2P

I^2P has not attracted as much attention as TOR so far. It is thus hard to judge its real qualities.

Unlike TOR, anonymity is not fully granted. Instead a pseudonymity is used.

In [24] an attack specific to I^2P is presented. As I^2Ps security model is chosen based on IP addresses, the authors propose to use several cloud providers in different B-Class networks. By selectively flooding peers, an adversary may extract statistical information. The paper proposes an attack based on the heuristic performance-based peer selection. The main criticism of the paper were that the peer selection may be influenced by an adversary enabling him to recover data on a statistical base.

MessageVortex does only allow a routing block builder to choose routes and amount of traffic. Due to the replay protection and the trust, we do not rely on any node, we show in [18] that attacks on this level are not possible.

4.1.4 Freenet

While Freenet is a pure distributed storage system it has many good features adapted by MessageVortex. Like in Freenet a MessageVortex node may deny to be the owner of a specific information unless the key for the respective ephemeral identity can be found on the system. As the key is only required for building routing blocks but not for message assembly and sending, this makes it a valuable feature comparable to the deniability of Freenet.

5 Conclusion

The MessageVortex protocol outlined in the previous sections does not solve all privacy issues which might arise. Furthermore, it is complicated to implement and involves a considerable amount of book keeping at runtime which is left to the sender of a message and the mixing nodes.

On the positive side, we have a new protocol which addresses privacy in a holistic approach leaving very little attack surface. If handled with appropriate care by the sender and receiver, the protocol allows a sender-controlled, high degree amount of anonymity. Message paths are diagnosable, may be built redundant and do not build on the trust of any third party systems including all involved mixes except the sender's and receiver's one. Even closed group communication or broadcasting to multiple identities involving a specific subset of mixes is possible if desired by the sender.

In [18] we show that the protocol is very secure. It is hard to block as messages may be redundant, hard to identify as messages are covered within message flows which may not be blocked without huge impact on existing systems. It is hard to apply censorship in a real world scenario as messages are extremely hard to detect.

MessageVortex has some flaws which must be outlined. We always considered an algorithmic censorship. If human censorship is applied, we must assume that at least some of the messages are being identified as possible MessageVortex messages. If we assume a white-listing, human, censoring adversary (everything which is not identified by a human as compliant is censored) we must conclude that at least some messages will fail to be delivered.

Some of the participating transport nodes may be identified and blocked. This may be compensated with redundancy in message transmission.

Messages transported by MessageVortex generate huge amounts of decoy traffic. Unlike other systems which control decoy traffic on a "per peer" base, MessageVortex does not dynamically reduce decoy traffic as decoy traffic is not identifiable. This results in a huge traffic overhead.

APPENDIX A
REED-SOLOMON FUNCTION

The origins of the Reed-Solomon code go back to [25]. The method described in this paper was however not applicable in all cases. The publications [26] and [27] describe a more practical solution whereas [28] brings up the similarity to the Reed-Solomon code with a $GF(2^{\omega})$.

Reed-Solomon is used for many applications today. One of the most well known application is a redundancy generator for RAID-6 like systems. It is able to generate multiple linearly independent equation systems to a given set of data $B_1 \dots B_{n-m}$ creating redundancy information $R_1 \dots R_m$. In a system with all blocks $\langle B_1 \dots B_{n-m}, R_1 \dots R_m \rangle$ any number k where $0 \leq k \leq m$ datablocks may be removed and the information contained in $\langle B_1 \dots B_{n-m} \rangle$ is still recoverable.

Traditionally the data and redundancy information is striped into blocks and distributed together with the redundancy information over all n storages. This is done to avoid data storages as bottleneck since a change to one data stripe in a stripe set results always in a change of the redundancy data on the other m redundancy storages. This would result in hot spots on redundancy information storages.

We use the Reed-Solomon function as redundancy generating function shown in Fig 2. Unlike in storage technology we encrypt each redundancy block and all data stripes individually. By doing so we make it impossible to recover the contained information without knowledge of the keys. All blocks do then contain the same amount of data. Given we have enough blocks and the corresponding keys we may rebuild the message.

At the same time the generating node is unable to tell what blocks belong to the true message path and what blocks are sent for decoy traffic only.

As our resulting blocks are encrypted with a stream or block cypher, we need to introduce some padding. The padding is applied before doing RS calculation. In the case of a stream cypher we need to pad so that the number of bytes is dividable by the number of data blocks. In the case of block cyphers we need to pad so that all resulting data blocks have exactly a size dividable by the block size. We achieve two goals by applying the padding before spliting the blocks. First we reduce overhead by adding only one instead of n paddings. Secondly, an unpadded block is much harder to brute force. Any resulting block to a key might be the right one as we no longer have padding to suggest that a decryption has been successful.

We defined to use Vandermonde matrices as outlined in [29] and in [30] for our redundancy calculations. For more information of the used GF-Fields and exact matrix building instructions see [18].

APPENDIX B
PSEUDO RANDOM NUMBER GENERATOR

Our PRNG used for this work is an xorshift+ generator. It is based on the XSadd PRNG [31] and passes the bigcrush PRNG test suite. It is a fast, xor based PRNG which has two internal 64 bit seed states s_0 respectively s_1 and is defined as follows:

$$x = s_0 \tag{17}$$

$$s_0 = s_1 \tag{18}$$

$$x = x \oplus (x \ll 23) \tag{19}$$

$$s_1 = x \oplus s_1 \oplus (x \gg 17) \oplus (s_1 \gg 26) \tag{20}$$

$$nextNumber = s_1 + s_0 \tag{21}$$

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

As the protocol is heavily dependent on security we have introduced everywhere at least one alternate algorithm which may be used if one of the choices may become a problem. In order to have a second choice for the PRNG we define the Blum-Micali PRNG as described in [32]. This PRNG is a cryptographically secure PRNG and is defined as follows:

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

$$x_{i+1} = g^{x_i} \mod p \tag{22}$$

APPENDIX C
ADVERSARY MODEL

We assume as an adversary a state-sponsored actor who has unlimited monitoring possibilities on the network layer within a limited geographic region (e.g. country). We furthermore assume available funding and capabilities to efficiently run a considerable number of nodes (not exceeding the number of 70%) within the network and harvest and combine all operations and content processed by these nodes.

This means he is at least capable of analysing traffic by algorithms and may disrupt any unwanted traffic.

The adversary is capable of generating any traffic anywhere within the message flow.

This is an adversary model which goes far beyond any model we have encountered in any other scientific approach.

Assumptions in this part are mostly derived from [23] and extrapolated. In this paper researchers describe GFC (Great Firewall of China) as a profiling firewall fingerprinting traffic and blocking specific socket addresses when positively identified as TOR nodes.

C.1 Detection

Any adversary may use detection schemes to detect traffic. This may be used to apply censorship later. He is capable of analysing up to 1% of the total transfer volume on an in-depth algorithmic base and 50% based on simple context-less rules.

C.2 Censorship

We assume that an adversary is accepting considerable economic damage but not the downfall of its own economy as a whole. An active adversary may apply algorithmic censorship based on the detection constraint (C.1) on a large scale.

C.3 Information Retrieval

We assume an adversary to be interested in all sorts of information available around messages. We consider the triple "sender", "receiver", and "content" as the most valuable set of information for an adversary. Other important informations are:

- frequency of message interchange
- message size

For the system we consider the following information as interesting for an adversary (descending order):

- Routing nodes
- Transporting nodes
- Routing operations
- Routing (ephemeral) identities
- Routing volumes

Acknowledgments

The authors would like to thank their families for being so patient with them, and all people which took the time to review the paper, criticised it, and their questions outlining the papers weaknesses.

References

[1] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003, pp. 2–15. [Online]. Available: http://mixminion.net/minion-design.pdf

[2] C. Gülcü and G. Tsudik, "Mixing E-mail with Babel," in *Proceedings of the Network and Distributed Security Symposium - NDSS '96*. IEEE, Feb. 1996, pp. 2–16. [Online]. Available: http://citeseer.nj.nec.com/2254.html

[3] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster Protocol — Version 2," IETF Internet Draft, Jul. 2003. [Online]. Available: http://tools.ietf.org/pdf/draft-sassaman-mixmaster-03.pdf

[4] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, Aug. 2004. [Online]. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA465464

[5] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, Jul. 2000. [Online]. Available: http://freehaven.net/doc/berk/freehaven-berk.ps

[6] S. Goel, M. Robson, M. Polte, and E. G. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University, Ithaca, NY, Tech. Rep. 2003-1890, Feb. 2003. [Online]. Available: http://www.cs.cornell.edu/People/egs/papers/herbivore-tr.pdf

[7] L. Overlier and P. Syverson, "Locating Hidden Servers," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006. [Online]. Available: http://tor-svn.freehaven.net/anonbib/cache/hs-attack06.pdf

[8] M. V. Barbera, V. P. Kemerlis, V. Pappas, and A. Keromytis, "CellFlood: Attacking Tor onion routers on the cheap," in *Proceedings of ESORICS 2013*, Sep. 2013. [Online]. Available: http://www.cs.columbia.edu/~vpk/papers/cellflood.esorics13.pdf

[9] A. Biryukov, I. Pustogarov, and R. P. Weinmann, "TorScan: Tracing long-lived connections and differential scanning attacks," in *Proceedings of the European Symposium Research Computer Security - ESORICS'12*. Springer, Sep. 2012. [Online]. Available: http://freehaven.net/anonbib/papers/torscan-esorics2012.pdf

[10] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for tor hidden services: Detection, measurement, deanonymization," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013. [Online]. Available: http://www.ieee-security.org/TC/SP2013/papers/4977a080.pdf

[11] D. Chaum, "Untraceable electronic mail, return, addresses, and digital pseudonyms," *Communications of the ACM*, 1981. [Online]. Available: http://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/chaum81.pdf

[12] ——, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, pp. 65–75, 1988. [Online]. Available: http://www.cs.ucsb.edu/~ravenben/classes/595n-s07/papers/dcnet-jcrypt88.pdf

[13] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, Jul. 2000, pp. 46–66. [Online]. Available: https://freenetproject.org/

[14] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5: A protocol for scalable anonymous communication," *Journal of Computer Security*, vol. 13, no. 6, pp. 839–876, 2005. [Online]. Available: http://www.cs.umd.edu/projects/p5/

[15] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," *ACM Transactions on Internet Technology (TOIT)*, vol. 5, no. 2, pp. 299–327, 2005. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/2.pdf

[16] J. Fidrich, M. Goljan, and D. Hogea, "Steganalysis of jpeg images: Breaking the f5 algorithm," in *International Workshop on Information Hiding*. Springer, 2002, pp. 310–323. [Online]. Available: http://www.ws.binghamton.edu/fridrich/research/f5.pdf

[17] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable anonymous group messaging," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. ACM, 2010, pp. 340–350. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866346

[18] M. Gwerder, "Messagevortex – transport independent messaging anonymous to third parties," Dec. 2017, PhD thesis writing in progress.

[19] G. Danezis and A. Serjantov, "Statistical disclosure or intersection attacks on anonymity systems," in *Proceedings of 6th Information Hiding Workshop (IH 2004)*, ser. LNCS, May 2004. [Online]. Available: http://www.cl.cam.ac.uk/~aas23/papers_aas/PoolSDA2.ps

[20] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the tor network," DTIC Document, Tech. Rep., 2014.

[21] S. J. Murdoch and G. Danezis, "Low-cost traic analysis of Tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005. [Online]. Available: http://www.cl.cam.ac.uk/users/sjm217/papers/oakland05torta.pdf

[22] S. Chakravarty, A. Stavrou, and A. D. Keromytis, "Traffic analysis against low-latency anonymity networks using available bandwidth estimation," in *Proceedings of the European Symposium Research Computer Security - ESORICS'10*. Springer, Sep. 2010. [Online]. Available: http://www.cs.columbia.edu/~sc2516/papers/chakravartyTA.pdf

[23] P. Winter and S. Lindskog, "How the great firewall of china is blocking tor," in *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI 2012)*, Aug. 2012. [Online]. Available: https://www.usenix.org/system/files/conference/foci12/foci12-final2.pdf

[24] M. Herrmann and C. Grothoff, "Privacy implications of performance-based peer selection by onion routers: A real-world case study using i2p," in *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS 2011)*, Jul. 2011. [Online]. Available: http://freehaven.net/anonbib/papers/pets2011/p9-herrmann.pdf

[25] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, Jun. 1960. [Online]. Available: https://faculty.math.illinois.edu/~duursma/CT/RS-1960.pdf

[26] E. Karnin, J. Greene, and M. Hellman, "On secret sharing systems," *IEEE Transactions on Information Theory*, vol. 29, no. 1, pp. 35–41, 1983. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/1056621/

[27] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989. [Online]. Available: http://doi.acm.org/10.1145/62044.62050

[28] F. P. Preparata, "Holographic dispersal and recovery of information," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 1123–1124, 1989. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/42233/

[29] J. S. Plank, "A tutorial on reed-solomon coding for fault-tolerance in raid-like systems," University of Tennessee, Tech. Rep. CS-96-332, Jul. 1996. [Online]. Available: http://web.eecs.utk.edu/~plank/plank/papers/CS-96-332.html

[30] J. S. Plank and Y. Ding, "Note: Correction to the 1997 tutorial on reed-solomon coding," University of Tennessee, Tech. Rep. CS-03-504, Apr. 2003. [Online]. Available: http://web.eecs.utk.edu/~plank/plank/papers/CS-03-504.html

[31] G. Marsaglia *et al.*, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003.

[32] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984. [Online]. Available: http://dx.doi.org/10.1137/0213053

Martin Gwerder

Martin Gwerder was born 20. July 1972 in Glarus, Switzerland. He is currently a PhD student at the University of Basel.

After having concluded his studies at the polytechnic at Brugg in 1997, he did a postgraduate education as a master of business and engineering. Following that, he changed to the university track doing an MSc in Informatics at FernUniversität in Hagen.

While doing this, he steadily broadened his horizon by working for industry, banking, and government as an engineer and architect in security-related positions.

He currently holds a lecturer position for cloud and security at the University of Applied Sciences Northwestern Switzerland. His primary expertise is in the field of networking-related problems dealing with data protection, distribution, confidentiality, and anonymity.