Unobservable Messaging with MessageVortex

Martin Gwerder

Abstract—In this paper, we introduce an unobservable message anonymization protocol, named MessageVortex. It bases on the zero trust principle and a distributed peer-to-peer like architecture and avoids central aspects such as fixed infrastructures within a global network.
It scores over existing work by blending its traffic into suitable existing transport protocols, thus making it next to impossible to block it without significantly affecting regular users of the transport medium. No additional protocol-specific infrastructure is required in public networks and allows a sender to control all aspects of a message such as the degree of anonymity, timing, and redundancy of the message transport without disclosing any of these details to the routing or transporting nodes. The most recent RFC-draft describes the protocol in detail. This draft contains all the necessary information to build protocol nodes. More information and a prototype implementation are available under https://messagevortex.net/.

Index Terms—Computer Society, IEEE, IEEEtran, journal, LaTeX, paper, template.

1 Introduction

Almon Brown Strowger was the owner of a funeral parlor in St. Petersburg. He filed a patent on March 10th, 1891 for an "Automatic Telephone Exchange" [1]. This patent built the base for modern automated telephone systems. According to several sources, he was annoyed by the fact that the local telephone operator was married to another undertaker. She diverted potential customers of Mr. Strowger to her husband instead, which caused Almon B. Strowger to lose business. In 1922, this telephone dialing system, which is nowadays called pulse dialing became the standard dialing technology for more than 70 years until tone dialing replaced it.

This dialing technology enabled automatic messaging for voice and text messages (e.g., telex) up until today and is the foundation for current routed networks. These networks build the base for our communication-based Society these days and allow us to connect quickly with any person or company of our wish. We use these networks today as communication meaning for all purposes, and most of the people spend minimal thoughts on the possible consequences arising if someone puts hands on this communication.

Collected data may be used to judge upon our intentions and thus is not only confidential if we have something to hide. This problem has dramatically increased in the last years as big companies and countries started to collect all kinds of data and created the means to process them. It allows, supposedly, to judge peoples not only on what they are doing but as well, on what they did and what they might do. Numerous events in the present and past show that multiple actors, some of which are state-sponsored, collected data on a broad base within the Internet. Whether this is a problem or not may be disputable. Undisputed is that such data requires careful handling, and accusations should then base on solid facts. Unacceptable seems the use of "guesses" or "extrapolations" in most of the cases.

• *M. Gwerder was with the Institute of Mobile and Distributed Systems of the University of Applied Sciences of Northwestern Switzerland. E-mail: see http://www.messagevortex.net*

Manuscript received June 1, 2019; revised July 20, 2019.

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 900000 files in a secret archive (covering roughly 10% of the natural and juristic entities within Switzerland at that time). The Swiss Federal Archives (https://www.bar.admin.ch) documents more about the Fichenskandal.

Whistleblower Edward Snowden leaked a vast amount of documents. These documents suggested that such attacks on privacy are commonly made on a global scale. The documents leaked in 2009 and a significant number of journalists from multiple countries screened them (NRC). According to these documents, NSA infiltrated more than 50k computers with malware to collect classified or personal information. They furthermore infiltrated Telecom-Operators (mainly executed by British GCHQ) such as Belgacom to collect data and targeted high member of governments even in associated states (such as the mobile phone number of Germany's president and high ranking members of economy and finance).

This list of events shows that big players are collecting and storing vast amounts of data for analysis or possible future use. The list of events also shows that the use of this data has in the past been at least partially questionable. As a part of possible countermeasures, this work analyses the possibility of using state-of-the-art technology to minimize the information footprint of a person on the Internet.

We leave a large information footprint in our daily communication. On a regular email, we disclose everything in an "postcard" to any entity on its way. Even when encrypting a message perfectly with today's technology (S/MIME [2] or PGP [3]) it still leaves at least the originating and the receiving entity disclosed, or we rely on the promises of a third party provider which offers a proprietary solution. Even in those cases, we leak pieces of information such as "message subject", "frequency of exchanged messages", "size of messages", or "client being used". A suitable anonymity protocol has therefore not only a message to hide but additional attributes as well. It includes besides

the message itself, all metadata, and all the traffic flows. Furthermore, a protocol to anonymize messages should not rely on the trust of infrastructure other than the infrastructure under control of the sending or receiving entity. Trust in any third party might be misleading in terms of security.

Central infrastructure is bound to be of particular interest to anyone gathering data. It may furthermore allow manipulating the system or the data or the data flow. So, avoiding a central infrastructure is a good thing.

Leaving no information trail when sending information from one person to another is hard to achieve. Most messaging systems disclose at least the peer partners when sending messages. Metadata such as starting and endpoints, frequency, or message size are leaked in all standard systems even when encrypting messages.

Allowing an entity to collect data may affect senders and recipients of any information. Collection of vast amounts of data allows a potent adversary to build a profile of a person. Unlike in the past, the availability of this kind of information has been risen to a never known extend with the Internet.

An entity in possession of such Profiles may use them for many purposes. These include service adoption, directed advertising, or classification of citizens. The examples given above show that the effects of this data is not limited to the Internet but reaches us effectively in the real world.

The main problem of this data is that it may be collected over a considerable amount of time and evaluated at any time. It even happened that standard practices of the time are judged differently upon later. Persons may then be judged retrospectively upon these types of practice. This questionable type of judgment is visible in the tax avoidance discussion.

People must be able to control their data footprint. Not providing these means does effectively allow any country or a bigger player to ban and control any number of persons within or outside the Internet.

In this work, a new protocol is designed to allow message transfer through existing communication channels. These messages are next to unobservable for any third party. This unobservability does not only cover the message itself but all metadata and flows associated with it. We called this protocol "MessageVortex" or in short just "Vortex". The protocol is designed in such a way so that it is capable of using a wide variety of transport protocols. It is even possible to switch protocols while the messages are transferred. This behavior allows media breaches (at least on a protocol level) and makes analysis even harder.

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

1.1 About Unlinkability, Unobservability, Undetectability and Censorship resistance

From an academic point of view, achieving anonymity is relatively simple. All we need is a trusted party distributing the messages while making sure that no trace from the sender arrives at the recipient. Unlinkability, as defined in [4] is much harder to achieve. It requires that a specified attacker is unable to link a sender and recipients of a message. Even if a system provides unlinkable message transfer, it is most probable prone to denial of service and thus partial or full censorship. By introducing a global observer or infiltrating parts of the system, an attacker may gain insight into the messages transported by the system and thus leaking information.

So to be censorship resistant, a protocol requires many critical, unobvious properties. As outlined It should be undetectable from the outside. From within the system, we need to provide a mean to make it very hard to follow message flows or identify participants.

MessageVortex is a protocol providing censorship resistance under ideal circumstances. It does this using a rigid design from bottom up to provide the required properties. While being a protocol on its own, it uses many standard protocols. Partly to provide user-friendliness, but mostly to hide within the regular network flows. As such, a protocol requires to be undetectable on the network. A protocol all alone may not be undetectable as each protocol sends data over a network. This data is detectable. To be undetectable, a censorship-resistant protocol requires to be embedded undetectably in legit message flows. Such embedding is usually done either by side channel transmissions or by employing steganography. Our protocol may use both but will use typically the later.

1.2 Adversary Model

For our adversary, we need to assume someone of the capabilities of a state-sponsored actor. State-sponsored actors may work these days with allies but will not achieve dominance over all jurisdictional domains.

For our adversary model we assume the following properties:

- An adversary will have elaborated technical know-how to attack any infrastructure.
- An adversary may have the capability to monitor traffic at many points in most networks within a jurisdiction.
- An adversary may have the capability to modify routing information within a jurisdiction freely.
- An adversary may have the possibility to freely modify even cryptographically weak secured data where a single or a limited number of entities grant proof of authenticity or privacy.
- An adversary may have the possibility to inject or modify any data on the network of a jurisdiction.
- An adversary may create own nodes of a network. He may furthermore monitor their behavior and data flow to the maximum possible extent.
- An adversary may force a limited number of other non-allied nodes to expose their data to him.
- An adversary may have a similar access to resources as within its jurisdiction in a limited number of other jurisdictions.

We assume the following goals for an adversary:

- An adversary may want to disrupt non-authorized communication.

- An adversary may want to read any information passing throughout the internet.
- An adversary may want to build and conserve data about individuals or groups of individuals of their life. This data includes all activities in any part of their life regardless of their apparent fitness for any specific purpose or their correctness.

1.3 Notation

The theory in this document is heavily based on symmetric encryption, asymmetric encryption, and hashing. To use an uniformed notation we use $E^{K_a}(M)$ (where a is an index to distinguish multiple keys) for an encrypting function with a key K_a. This results in $\mathbf{M^{K_a}}$ for the encrypted message. If we are reflecting a tuple of information, we write in boldface. To express a concatenated set of information, we use angular brackets $\mathbf{L}\langle\mathbf{normalAddress}, \mathbf{vortexAddress}\rangle$. If we want to differentiate messages encrypted with multiple keys, we list the used keys as a comma-separated list in superscript $E^{K_b}\left(E^{K_a}(M)\right) = M^{K_a,K_b}$.

For a symmetric encryption of a message $\mathbf{M}$ with a key K_a resulting in $\mathbf{M^{K_a}}$ where a is an index to distinguish different keys. Decryption uses therefore $D^{K_a}(\mathbf{M^{K_a}}) = \mathbf{M}$.

As notation for asymetric encryption we use $E^{K_a^1}(\mathbf{M})$ where as K_a^{-1} is the private key and K_a^1 is the public key of a key pair K_a^p. The asymmetric decryption is noted as $D^{K_a^{-1}}(\mathbf{M})$.

For hashing, we do use $H(\mathbf{M})$ if unsalted and H^{S_a} if using a salted hash with salt S_a. The generated hash is shown as H_M if unsalted and $H_M^{S_a}$ if salted.

If we want to express what details contained in a tuple we use the the notation $\mathbf{M}\langle\mathbf{t}, \mathbf{MURB}, \mathbf{serial}\rangle$ respectively if encrypted $\mathbf{M^{K_a}}\langle\mathbf{t}, \mathbf{MURB}, \mathbf{serial}\rangle$.

$$
\begin{aligned}
\text{asymetric:} E^{K_a^{-1}}(\mathbf{M}) &= \mathbf{M}^{K_a^{-1}} \\
D^{K_a^1}\left(E^{K_a^{-1}}(\mathbf{M})\right) &= \mathbf{M} \\
D^{K_a^{-1}}\left(E^{K_a^1}(\mathbf{M})\right) &= \mathbf{M} \\
\text{symetric:} E^{K_a}(\mathbf{M}) &= \mathbf{M}^{K_a} \\
D^{K_a}\left(E^{K_a}(\mathbf{M})\right) &= \mathbf{M} \\
\text{hashing (unsalted):} H(\mathbf{M}) &= \mathbf{H}_M \\
\text{hashing (salted):} H^{S_a}(\mathbf{M}) &= \mathbf{H}_M^{S_a}
\end{aligned}
$$

In general Subscripts denote selectors to differentiate values of the same type and superscript denote relevant parameters to operations expressed. The subscripted and superscripted information may be omitted where not needed.

We refer to the components of a Vortex Message as follows:

$$
\begin{aligned}
\text{Prefix component:}\mathbf{PREFIX} &= D^{K_a^1}\left(\mathbf{P}^{K_a^{-1}}\right) = D(\mathbf{P}) \\
\text{Header component:}\mathbf{HEAD} &= D^{K_a^1}\left(\mathbf{H}^{K_a^{-1}}\right) = D(\mathbf{H}) \\
\text{Route component:}\mathbf{ROUTE} &= D^{K_a^1}\left(\mathbf{R}^{K_a^{-1}}\right) = D(\mathbf{R})
\end{aligned}
$$

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

To specify a random byte sequence of n octets generated by an PRNG of type t and seeded with s we use $R_t(s, n)$

2 The MessageVortex Protocol

2.1 General Design

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

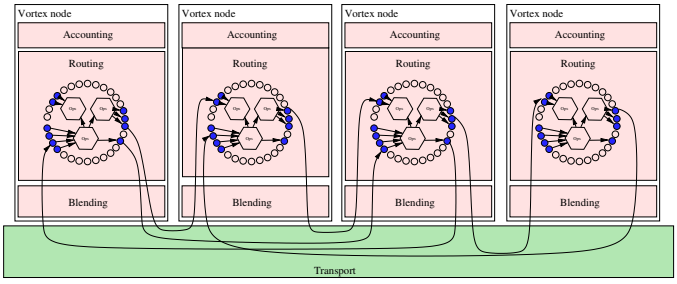

Fig. 1. A rough protocol outline of the MessageVortex protocol

The protocol handles messages which are passed by a transport protocol from node to node. The instructions how and when to pass a message is generated by the "routing block builder" (RBB). The RBB defines the path of the message, the type of blending on each nodes, and the operations applied to each part of the message in each node. To avoid collision of operations, an RBB has identities on each node with an assigned workspace. In these workspaces messages may be assembled, transformed, or decomposed with the aforementioned operations. Results are sent to other nodes. Due to the nature of the operations all messages passed on may be decoy traffic or "real messsage parts" (We proof that claim in section 3.1). A node will know that a message is destined for it when a message results in data in specific parts of the identities workspace.

We divide our protocol into four different layers, whereas only three are specific to the MessageVortex protocol. The lowest layer is the transport layer. As expressed earlier, dedicated protocols are easy to censor. Therefore we build our protocol on top of other suitable transport protocols.

The other Three layers are Vortex-specific and do not require any infrastructure on the Internet. We elaborate further on these layers in section 2.3 and section 2.4.

For our first tests, we used a custom transport layer allowing us to monitor all traffic quickly, and build structures in a very flexible way. This transport layer works locally with a minimum amount of work for setup and deployment. It furthermore works across multiple hosts in a broadcast domain. The API may be used to support almost any kind of transport layer. After that, we focused on the protocols identified suitable as transport protocols:

- SMTP
- XMPP

For the prototype, we have implemented an SMTP transport agent and the respective blending layer.

The blending layer is taking care of multiple problems:

- Translating the message into the transport format
  This translation includes jobs such as embedding a message as encoded text, as a binary attachment or hide it within a message using steganography.
- Extract incoming messages from transport
  Identify incoming messages containing a possible block and extract it from the message.
- Do housekeeping on the storage layer
  Access protocols POP and IMAP require that messages are deleted from time to time in order to stay below the sizing quotas of an account.

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

1) Log arrival time (in UTC) on the transport layer.
2) Extract possible blocks.
3) Apply decryption on a suspected header block.
4) Validate the header block using the accounting layer.
5) Process header requests (if any)
6) Extract and decrypt subsequent blocks.
7) Pass extracted blocks and information to the routing layer.

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

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

The routing layer receives the message blocks in a decrypted and authorized form from the blending layer. The routing layer then assembles all information of an identity and makes executes the accepted operations using the available data.

Tasks of the transport layer are:

- Build structure representing the block building and the appropriate block IDs.
- Schedule all Routing blocks for processing in an priority queue.
- Authorize all routing blocks ready for processing with the calculated block sizes.
- Process blocks.
- Send prepared building blocks to the Blending layer.

2.1.1 Identities and Keys

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

2.1.2 Peer key

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

2.1.3 Header Key

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

The Identity key is a static, asymmetric keypair existing on a per user base used to sign messages or encrypt symmetric keys. We refer here as a user any peer participating in the message stream. Every user participating in the Vortex network requires at least one key pair. Depending on the use-case (e.g., unlikable signatures or scalable security) a user may use multiple key pairs at the same time.

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

2.2 Messages

The outline in figure 2 is a simplified view of the message block of MessageVortex.

A Vortex message is a message passed from one router to another this message may or may not contain any valuable information. Binary data represents the message content in the transfer protocol. Every router may decide for himself on the support of algorithms and embedding mechanisms.

The block structure of a Vortex message is as follows:

- padding
  - Encrypted message key k_{peer}.
    Contains symmetrical key for decryption of follow up header information and payload blocks. The key is encrypted with the receiving host's public key. The key is known to the RBB, the message sending node and the message receiving node. No other node has knowledge of the key.
- header block
  - Identity and "proof of work" information
    This information contains requests, proof-of-work sections, and a header key.
  - Replay protection information
    This information allows a node to identify replayed Messages even if the payload of the content has been modified.
  - Forward secret information
    This information allows a node to identify VortexMessages where tampering occurred by recombining blocks of multiple messages.
  - (optionally) Proof-of-work information
    This information allows a sending node to fulfill a proof of work requirement raised due to a previously sent request.
- Routing blocks (encrypted with message key)
  - Next hop timing instructions
  - Next hop routing blocks (encrypted)
  - Next hop header
  - Message build instructions.
  - Next hop header key and spec.

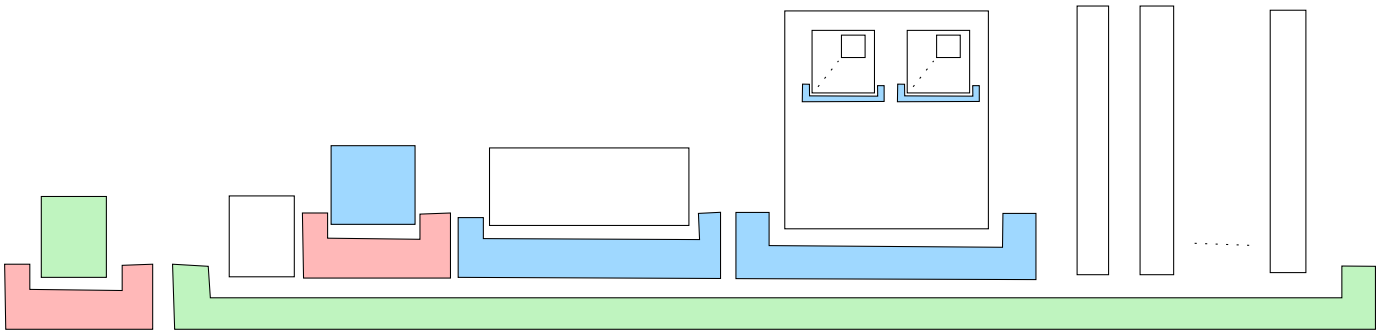

Fig. 2. Simplified message outline

  - Next hop blending instructions.
- Encrypted payload blocks (encrypted with header key)
  - Payload chunk blocks

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

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

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

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

2.3 Transport and Blending Layer

The tuple of Transport and Blending layer builds the main core functionality when it comes to the undetectability property. In order to avoid transport protocol misuse and unintentional exit nodes of the protocol the original RBL has no control over the transported content except for the hidden VortexMessage and how it is embedded. This loads the burden of sensible clear text payload generation to blending layer. It is relatively easy to generate a credible clear text message to pass an automated testing engine. This can be easily proven by looking at the effectivity of junk mail filters. These filters have tremendous problems continuosly adapting to the new types of unsolicited bulk emails (UBE).

Things do however drastically change if taking a human censor into account.

2.3.1 *Applied Steganography and the Dead Parrot*

A human censor is able to take very complex information into account when it comes to analyses of message content. He is able to not only analyze a message for its content, but he may also see the message in the context of other messages. In [5] is expressed that it is easy for a human to determine decoy traffic as the content is easily identifiable as generated content. While this is true for the very general case, there is a possibility here to generate “human-like” data traffic to a certain extent. As an adversary may not assume that his messages are replied to, the problem does not boil down to a Turing test. It remains on the level of a “passive observer Turing test”, enabling the potential nodes to choose their messages and the replies generated to them.

The easiest approach would have been to give a routing block builder the means of controlling the decoy message content. While such a possibility would be easy it would enable a routing block builder to use the node as a “exit node” from the system. Blackmailing messages ccould be sent through the system to a non-participating member and leak at the same time the presence of a routing node. To deny this possibility we shifted the ability to the routing node.

The message itself is binary and as such there are only limited possibilities to hide the VortexMessage within the transport protocol. We decided to typically use attachments or attachment-like structures. Within the attachments we currently support two types of embedding: plain and steganographic embedding.

Plain embedding means that we insert a sequence of blocks into a normal message. This is typically done wihin files with a weaker structure and a high entropy (such as an MP3 encoded file). While this is very hard to detect for a machine, it becomes immediately suspicios for a human cesor. A human censor would detect the presence of a payload which does not make any sense.

For steganographc embedding we decided to go for F5 [6]. It is a reasonably well-researched algorithm which

attracted many researchers. The original F5 implementation had a detectable issue with artifacts [7] caused by the recompression of the image. This issue was caused only due to an issue in the reference implementation, and the researchers have provided a corrected reference implementation without the weakness. Like always the type of embedding may be specified and replaced upon request.

2.4 Routing and Accounting

Each routing layer does the anonymization operations. These operations should make sure that no message may be identified unless destined for the current node.

2.5 Header Requests

There are several header requests available to the protocol mainly supporting the message transfer. There are header requests for creating ephemeral identities on a node , and for raising a message or size quota. Additionally, there are some helper request for querying quotas of an known ephemeral identity, and for querying the capabilities of a node.

Some of the requests may be replied with the requirement of providing a "proof of work" in order to be fulfilled. A node is free to not reply to a request or reject it depending on its privacy needs and the current usage of resources.

2.6 The Core: Operations

We differentiate three types of operations:

- Splitting and merging of chunks
- Encryption and decryption of chunks
- Redundancy calculations carried out on chunks

The first two Operations don not really provide a high level of unlinkability as they do allow analysis such as hotspot analysis and produce continuously inclining, steady or declining message sizes depending on the type of use. The third operation however adds a whole lot of new possibilities in conjunction with the other two.

2.6.1 Splitting and Merging

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

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

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

$$split(f, pb_1) = \langle pb_1, pb_2 \rangle \quad (1)$$
$$pb_1.startsWith(pb_2) \quad (2)$$
$$pb_1.endsWith(pb_3) \quad (3)$$
$$len(pb_2) = floor(len(pb_1) \cdot f) \quad (4)$$
$$len(pb_1) = len(pb_2) + len(pb_3) \quad (5)$$

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

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

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

$$merge(pb_1, pb_2) = pb_3 \quad (6)$$
$$pb_3.startsWith(pb_1) \quad (7)$$
$$pb_3.endsWith(pb_2) \quad (8)$$
$$len(pb_3) = len(pb_1) + len(pb_2) \quad (9)$$

2.6.2 Encryption and Decryption

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

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

The operation follows the following rules (please note section 1.3 for notation):

$$encrypt(pb_1, spec, k) = pb_2 \quad (10)$$
$$pb_2 = E^{K_a}_{spec}(pb_1) \quad (11)$$
$$len(pb_2) \geq len(pb_1) \quad (12)$$

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

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

The operation follows the following rules (please note section 1.3 for notation):

$$decrypt(pb_1, spec, k) = pb_2 \quad (13)$$
$$pb_2 = D^{K_a}_{spec}(pb_1) \quad (14)$$
$$len(pb_2) \leq len(pb_1) \quad (15)$$

2.6.3 Redundancy Operations

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

The Operation itself is shown in 3. It may be subdivided into the following operations:

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

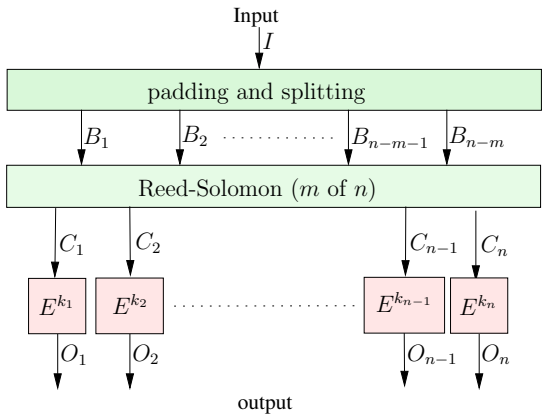

Fig. 3. Outline of the addRedundancy operation

The padding is not a standard padding from encryption. The reason for this lies in the properties required in the padding. These properties were:

- The padding must not leak whether the rebuild cycle was successful or not.
- The padding should only reveal true message size if a message was successfully rebuilt.
- The anyone knowing the routing block content and the transmitted message must be able to predict any treated block including all padding bytes
- The padding must work with any size padding space

The padded value $\mathbf{X}$ for a function $RS_{\text{m of n}}$ and an encryption block size of k is calculated as follows:

$$i = len(\mathbf{M}) \quad (16)$$
$$e = k \cdot n \quad (17)$$
$$l = \left\lceil \frac{i+4+C2}{e} \right\rceil \cdot e \quad (18)$$
$$p = i + C1 \cdot l \pmod{2^{32}} \quad (19)$$
$$\mathbf{X} = \langle p, \mathbf{M}, R_t\,(s, l-i)) \rangle \quad (20)$$

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

To reverse a successful message recovery information the of a padded block $\mathbf{X}$ we calculate the original message size by extracting p and doing $len(\mathbf{M}) = p - C2 \pmod{len(\mathbf{X})}$

2.7 Usage of the Protocol

First a sending node collects either a set of nodes and keys it wants to use and creates identities on these nodes using header requests. Then the sender creates a routing block containing all the routing instructions (hops and operations). Alternatively a sender may use a premanufactured routing block for the specified target. This routing block is then concatenated to a message and passed to the locally running routing node. From there the message is routed as defined in the routing block. An example for such a route is shown in Fig. 1.

3 Discussion of the Results

We first focus on the protocol itself to show the strength an weaknesses of the protocol. After that we focus on the dynamic part and see what type of data may be collected when considering not only the protocol but the whole message flow. We then present guidelines for different jurisdictional types.

We especially highlight adversary in an environment where the participation as a MessageVortex route is a criminal act and highlight some additional constraints applying in such situations.

3.1 Static Protocol Analysis

3.1.1 Endpoint Operation

3.2 Dynamic Protocol Analysis

3.2.1 Bootstrapping of Addresses and Identities

3.2.2 Discovery of Peer Nodes

Besides attacking the message content, attacking the routing nodes is an option for an adversary in a jurisdiction where operation of such a node is a criminal act. The presence of routing nodes is a valuable information for a global observer narrowing down the information regarding data to be analyzed.

3.2.3 Findings based on adversary environment

In environments containing only global observers and no jurisdictional constraints regarding the technology, a VortexNode may disclose its presence. This means a

3.3 The Missing Links

A lot still needs to be done. For this protocol to be of any use, a user-friendly implementation is required. The current implementation works as a prototype for academic research but is far beyond from being user friendly. Too many chices are left to the user currently. The new implementation must provide very good censorship resistance while providing easy to use recipes for message transfer.

For the traffic to be truly undetectable AIs must generate meaningful conversation between agents. This conversation does not necessarily boil down to a turing test. It is sufficient that two blending layer are capable to set up a communication which is undestinguishible from a normal human communication. As an adversary is typically not able to generate own traffic without exposing the probing activity and a blender is not required to such probes an attacker is very limited.

4 Conclusion

Creating a possibly censorship resistant protocol is already hard. Analysis showed that even when a protocol is crafted with great care braking unobservability is far easier than doing it right. At the same time we were able to present a system which requires an unmatched amount of observation and calculation power to be broken.

gwm
May 8, 2019

References

[1] A. B. Strowger, "Automatic telephone-exchange," United State Patent Oice Patent 447 918, 3 10, 1891.

[2] N. Freed and N. Borenstein, *RFC2045 Multipurpose Internet Mail Extensions; (MIME) Part One: Format of Internet Message Bodies*. IETF, 1996. [Online]. Available: http://tools.ietf.org/pdf/rfc2045.pdf

[3] M. Elkins, *RFC2015 MIME Security with Pretty Good Privacy (PGP)*. IETF, 1996. [Online]. Available: http://tools.ietf.org/pdf/rfc2015.pdf

[4] A. Pfitzmann and M. 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, Aug. 2010, v0.34. [Online]. Available: http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf

[5] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013. [Online]. Available: http://www.cs.utexas.edu/~amir/papers/parrot.pdf

[6] A. Westfeld, "F5 - a steganographic algorithm," *none*, vol. none, 2002. [Online]. Available: http://www.ws.binghamton.edu/fridrich/research/f5.pdf

[7] J. A. Briffa, H. G. Schaathun, and A. W. A. Wahab, "Has f5 really been broken." [Online]. Available: https://pdfs.semanticscholar.org/4d6c/d9d7e3a419ea74a4a363a36fcc674e89ecc7.pdf

Martin Gwerder

Martin Gwerder was born 20. July 1972 in Glarus, Switzerland. He is currently a doctoral Student at the University of Basel. After having concluded his studies at the polytechnic at Brugg in 1997, he did a postgraduate study as a master of business and engineering. Following that, he changed to the university track doing an MSc in Informatics at FernUniversität in Hagen. While doing this he constantly broadened his horizon by working for industry, banking and government as engineer and architect in security related positions. He currently holds a lecturer position for cloud and security at the University of Applied Sciences Northwestern Switzerland. His main expertise lays in the field of networking related problems dealing with data protection, distribution, confidentiality and anonymity.