Mixminion: Design of a Type III Anonymous Remailer Protocol

George Danezis
University of Cambridge
george.danezis@cl.cam.ac.uk

Roger Dingledine and Nick Mathewson
The Free Haven Project
{arma,nickm}@freehaven.net

Abstract

We present Mixminion, a message-based anonymous remailer protocol with secure single-use reply blocks. Mix nodes cannot distinguish Mixminion forward messages from reply messages, so forward and reply messages share the same anonymity set. We add directory servers that allow users to learn public keys and performance statistics of participating remailers, and we describe nymservers that provide long-term pseudonyms using single-use reply blocks as a primitive. Our design integrates link encryption between remailers to provide forward anonymity. Mixminion works in a real-world Internet environment, requires little synchronization or coordination between nodes, and protects against known anonymity-breaking attacks as well as or better than other systems with similar design parameters.

1. Overview

Chaum first introduced anonymous remailers over 20 years ago [7]. The research community has since introduced many new designs and proofs [1, 14, 16, 19, 28, 29], and discovered a variety of new attacks [3, 5, 6, 9, 23, 35]. But because many of the newer designs require considerable coordination, synchronization, bandwidth, or processing resources, deployed remailers still use Cottrell's Mixmaster design from 1994 [8, 26]. Here we describe Mixminion, a protocol for asynchronous, loosely federated remailers that maintains Mixmaster's flexibility while addressing the following flaws:

- **Replies:** Mixmaster does not support replies or anonymous recipients — people who want these functions must use the older and less secure Cypherpunk Type I remailer design [31], which is vulnerable to replay attacks. We introduce a new primitive called a *single-use reply block* (SURB), which makes replies as secure as forward messages. Furthermore in Mixminion the remailers themselves cannot distinguish reply messages from forward messages. We also describe how to use these SURBs to securely build higher-level systems such as nymservers. By integrating reply capabilities into Mixminion, we can finally retire the Type I remailer network.

- **Forward anonymity:** Mixmaster uses SMTP (normal mail) for transport. We use TLS over TCP for link encryption between remailers and use ephemeral keys to ensure forward anonymity for each message. Link encryption also blocks many active and passive attacks on the communication links.

- **Replay prevention and key rotation:** If an adversary records the input and output batches of a mix and then replays a message, that message's decryption will remain the same. Thus an attacker can completely break the security of the mix-net [7]. Mixmaster 2.0 offered replay prevention by keeping a list of recent message IDs — but because it expired old entries to keep the list short, the adversary simply has to wait until the mix has forgotten a message and replay it. Newer versions of Mixmaster keep a replay cache and also discard messages more than a certain number of days old. To block timestamp attacks, clients randomly add or subtract a few days from the timestamp. But this approach may still be open to statistical attacks; see Section 5.4. Mixminion instead counters replays by introducing key rotation: a message is addressed to a given key, and after the key changes no messages to the old key will be accepted, so the mix can forget about all the messages addressed to old keys. The number of IDs a node needs to remember between key rotations is not too great a burden.

- **Exit policies:** Exit abuse is a serious barrier to wide-scale remailer deployment: most Internet Service Providers (ISPs) do not tolerate systems that potentially deliver hate mail, etc. Mixminion provides a consistent mechanism for each node to specify and advertise an exit policy. We further describe a protocol which allows recipients to opt out of receiving mail from remailers, but at the same time makes it difficult for an adversary to deny service to interested recipients.

- **Integrated directory servers:** Mixmaster uses several *ad hoc* approaches to distribute information about remailer availability, performance, and keys. But the fact that users and remailers operate with different information introduces *partitioning* attacks. Mixminion uses a small group of synchronized redundant directory servers to provide uniform information about the network.
- **Dummy traffic:** Cottrell briefly mentions dummy messages in [8], but they are not part of the specification [26]. Mixminion uses a simple dummy policy for now, but because we use our own transport, we support many link padding and dummy traffic schemes.

We review mixes and mix-nets in Section 2, describe our goals and assumptions in Section 3, and then address the above list of improvements in Sections 4-7. We then summarize how our design stands up to known attacks, and conclude with a list of open problems.

2. Background

Chaum introduced the concept of using relay servers, or *mixes*, for anonymous communications [7]. Each mix has a public key which senders use to encrypt messages to it. The mix accumulates a batch of these encrypted messages, decrypts them, and delivers them. Because a decrypted output message looks nothing like the original encrypted input message, and because the mix collects a batch of messages and then sends out the decrypted messages in a rearranged order, an observer cannot learn which incoming message corresponds to which outgoing message. Chaum showed the security of a mix against a *passive adversary* who eavesdrops on all communications but is unable to observe the reordering inside the mix. Pfitzmann fixed a weakness in Chaum's original scheme based on the properties of raw RSA encryption [32].

However, trusting a single mix is dangerous: the mix itself could be controlled by an adversary. Therefore users send their messages through a series of mixes: if some of the mixes are honest (not run by the adversary), some anonymity is preserved. In some schemes, such as Mixmaster [26] and Babel [14], the sender chooses the mixes that make up her message's path. Specifically, when Alice wants to send an anonymous message to Bob through mixes M_1, M_2, and M_3, she encrypts her message successively with the public keys of the mixes in reverse order. She includes routing information at each hop, so that each mix M_i receives the address of M_{i+1} along with the message intended for M_{i+1} (all encrypted under M_i's public key).

A mix network where Alice chooses her route freely from all mixes is called a *free-route* network. Another approach is a *cascade* network, where senders choose from a set of fixed paths through the mix-net. Cascades can provide greater anonymity against an adversary who owns many mixes [6], but they are also more vulnerable to blending attacks such as trickle or flooding attacks [36]. Further, cascade networks arguably have lower maximum anonymity because the number of people Alice can hide among (her *anonymity set*) is limited to the number of messages the weakest node in her cascade can handle. In a free-route network, larger anonymity sets are possible because no single mix acts as a bottleneck: many mixes handle traffic in parallel as messages traverse the network. Mix cascade research includes real-time mixes [18] and web mixes [4].

More complex designs use zero-knowledge proofs and stronger assumptions to guarantee delivery or to detect and exclude misbehaving participants. These include flash mixes [16], hybrid mixes [17, 29], and provable shuffles [13, 28]. The properties of these designs are appealing, but they are often impractical since they assume fairly strong coordination and synchronization between the mixes and impose a heavy computational and communication overhead.

Some mix-net designs allow recipients to construct *reply blocks* that allow others to send messages to them without knowing their identities. A reply block contains only the routing portion of a message; the actual contents are appended by the user who eventually sends a message to the recipient. In this case the contents are effectively *encrypted* at each step in the path rather than decrypted. The recipient knows all the keys used in the reply block and can peel off all the layers of encryption when the message arrives. Such a design was first introduced by Chaum [7] and later extended in Babel [14]. However, Babel's replies are indistinguishable from forward messages only by passive observers — the mix nodes can still tell them apart. Babel's reply addresses are also multiple-use, making them less secure than forward messages due to replay vulnerabilities.

The first widespread public implementations of mixes were produced by contributors to the Cypherpunks mailing list. These "Type I" *anonymous remailers* were inspired both by the problems surrounding the `anon.penet.fi` service [15], and by theoretical work on mixes. Hughes wrote the first Cypherpunk anonymous remailer [31]; Finney followed closely with a collection of scripts that used Phil Zimmermann's PGP to encrypt and decrypt remailed messages. Later, Cottrell implemented the Mixmaster system [8, 26], or "Type II" remailers, which added message padding, message pools, and other mix features lacking in the Cypherpunk remailers.

2.1. Known attacks against mix-nets

Attacks against mix-nets aim to reduce the anonymity of users by linking anonymous senders with the messages they send, by linking anonymous recipients with the messages they receive, or by linking anonymous messages with one another [35]. Attackers may trace messages through the network by observing network traffic, compromising mixes, compromising keys, delaying messages so they stand out from other traffic, or altering messages in transit. They may learn a given message's destination by flooding the network with messages, replaying multiple copies of a message, or shaping traffic to isolate the target message from other unknown traffic. Attackers may discourage users from using honest mixes by making them unreliable. They may analyze intercepted message text to look for commonalities between otherwise unlinked senders. Finally, even if all other attacks are foiled, a passive adversary can mount a long-term *intersection attack* to correlate the times at which senders and receivers are active [6].

We discuss each of these attacks in more detail below, along with the aspects of the Mixminion design that provide defense. We provide a summary of the attacks and our defenses against them in Section 9.

3. Design goals and assumptions

Mixminion brings together the current best practical approaches for providing anonymity in a batching message-based free-route mix environment. We do not aim to provide a low-latency connection-oriented service like Freedom [37] or Onion Routing [39]: while those designs are more convenient for common activities like anonymous web browsing, their low latency necessarily implies smaller anonymity sets than with slower, message-based services. Indeed, we intentionally restrict the set of options for users: we provide only one cipher suite and we avoid extensions that would help an adversary partition the anonymity set. These assumptions lead to the following design goals:

First of all, the system must be *simple to deploy*. Past systems have never found it easy to get a reliable group of mix operators to run long-lived servers. Mixminion must add as few technical barriers as possible. Thus our protocol uses clock synchronization only to notice when a mix's key has expired, achieves acceptable performance on commodity hardware, requires little coordination between servers, and can automatically handle servers joining and leaving the system.

Furthermore, the system must be *simple for clients*. Because software adoption has also been a barrier to past systems, we attempt to make the requirements for senders and receivers as low as possible. Thus, only users who receive anonymity from the system must run special software – that is, users should be able to receive messages from anonymous senders and send messages to anonymous recipients with a standard email client. (Non-anonymous recipients receive messages via e-mail; non-anonymous senders using reply blocks send messages via e-mail gateways.) Users must also be able to send and receive anonymous messages using only commodity hardware. Finally, although users with persistent network connections are necessarily more resistant to intersection attacks than users with intermittent connections, the system must offer the latter users as much anonymity as possible.

We choose to *drop packet-level compatibility with Mixmaster* and the Cypherpunk remailer systems in order to provide a simple extensible design. We can retain minimal backwards compatibility by "remixing" Type II (Mixmaster) messages inside Type III (Mixminion) messages, thus increasing anonymity sets in the Type III network. (A Type II message traveling between backward-compatible Type III remailers is encrypted to the next remailer in the chain using its Type III key, and sent as a Type III encrypted message. The recipient decrypts it to reveal the Type II message.)

For our *threat model*, we assume a well-funded adversary who can observe all traffic on the network; who can generate, modify, delete, or delay traffic on the network; who can operate mixes of its own; and who can compromise some fraction of the mixes on the network. Our adversary tries to link senders and receivers, to identify the sender or receiver of a given message, or trace a sender forward (or a receiver backward) to its messages.

The Mixminion design tries to make it as hard as possible for an adversary observing the network to gain any additional information about communicating partners beyond its *a priori* belief. It does this by providing very little information to outside observers, and intermediate nodes, to avoid intersection attacks. In particular, even intermediary nodes are not aware of the actual route length (which can be as long as 32 hops) or their position in the network. Furthermore, the processing for replies is exactly the same as for normal messages, and it is therefore difficult to partition the anonymity sets by distinguishing between them.

4. The Mixminion Mix-net Design

Mixminion uses a free-route mix-net just like Mixmaster [26] and Babel [14]. Mixminion's principal difference from earlier mix-net designs is the mechanism it uses to support reply messages with the same processing machinery as forward messages, while at the same time resisting the attacks described above.

Mixminion does not implement reusable reply blocks, such as those in the Cypherpunk remailer and in Babel. They are convenient, but they pose a security risk – by their very nature they let people send multiple messages through them. An attacker can use this property to trace the recipient's path: if two incoming batches both include a message to the same reply block, then the next hop must be in the intersection of both outgoing batches. To prevent these replays, Mixminion provides only *single-use* reply blocks (SURBs). Since replies may be rare relative to forward messages, and thus much easier to trace, the Mixminion protocol makes reply messages indistinguishable from forward messages even for the mix nodes. Thus forward and reply messages can share the same anonymity set.[1]

Mixminion's reply model requires anonymous recipients to keep one secret for each nym they maintain. The final header of the SURB includes a seed (symmetrically encrypted to that nym's secret), from which the recipient can derive all the secrets needed to strip the layers of encryption from the received message. The recipient keeps a separate secret for each nym in order to block attacks to link the nyms. For example, if Alice is talking to Bob and Charlie and guesses they are the same person, she might reply to Bob's mail using Charlie's reply block; if Bob responds as normal, her guess would be confirmed.

The rest of this section describes the mechanism for secure replies, its integration with the normal sender anonymous message delivery, and how we defeat tagging-related attacks.

4.1. Recipient anonymity and indistinguishable replies

Mixminion allows Alice to send messages to Bob in one of three ways:

1. **Forward** messages where only Alice remains anonymous.
2. **Direct Reply** messages where only Bob remains anonymous.
3. **Anonymized Reply** messages where Alice *and* Bob remain anonymous.

The fact that forward messages are indistinguishable from replies, however, makes it more difficult to prevent *tagging attacks*. Since the author of a SURB cannot predict the message that will be attached to it, a hash of the entire message cannot be included in the SURB. Therefore, since we choose to make forward messages and replies indistinguishable, we cannot include hashes for forward messages either. Tagging attacks, and our approach to preventing them, are discussed in more detail in Section 4.2.

Messages are composed of a header section and a payload. We divide a message's path into two *legs*, and split

[1]Note that replies are still weaker than forward messages: an adversary can successively force intermediate mixes to reveal the next hop of the reply block until its originator is reached.

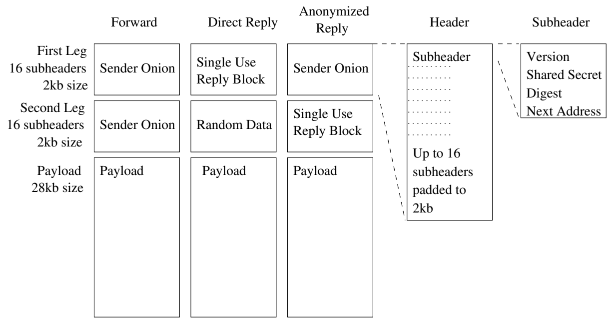

Figure 1. Header configurations for different anonymity functions

the header section correspondingly into a main header and a secondary header. Each header is composed of up to 16 subheaders, one for each hop along the path. Each subheader contains a hash of the remainder of its header as seen by that mix, so we can do integrity-checking of the path (but not the payload) within each leg. Each subheader also contains a master secret, used to derive a symmetric key for decrypting the rest of the message. To make sure that the hash matches even though each hop must repad the header to maintain constant message length, we need to add predictable padding to the end of the header: the mix appends an appropriate number of zero bits to the header after message decryption, and decrypts those also. A security proof for a simplified version of this approach is given in [25].

Each subheader also includes the address of the next node to which the message should be forwarded, along with its expected signature (identity) key fingerprint — the mix refuses to deliver the message until the next hop has proved its identity.

For forward messages, Alice provides both legs. For anonymous replies, Alice uses Bob's reply block as the second leg, and generates her own path for the first leg. To send a direct reply, Alice can use an empty second leg, or send the reply block and message to a mix that can wrap them for her. Figure 1 illustrates the three options.

When Alice creates her message, she encrypts the secondary header with a hash of her payload (as well as the usual layered onion encryptions). Alice's message traverses the mix-net as normal (every hop pulls off a layer, verifies the hash of the current header, and puts some junk at the end of the header), until it gets to a hop that is marked as a *crossover point*. This crossover point performs a "swap" operation: it decrypts the secondary header with the hash of the current payload, and then swaps the two headers. The swap operation is detailed in Figure 2 — specifically, the normal operations done at every hop are those above the dotted line, and the operations performed only by the crossover point are those below the dotted line. We use

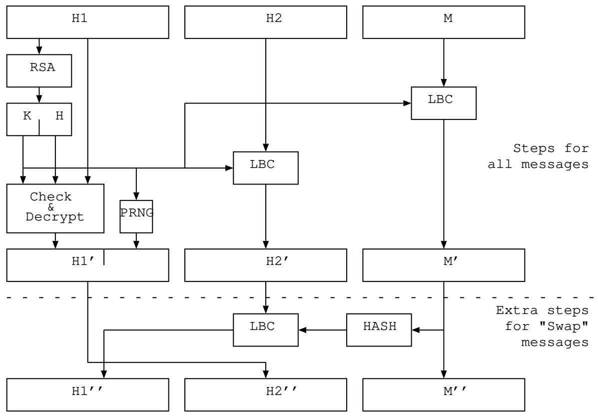

Figure 2. Operations performed by the "swap" method

a keyed encryption primitive, labeled "LBC" (for Large-Block Cipher), to encrypt the second header and the payload. This primitive needs to have certain properties:

- The LBC operation must preserve length.
- Without knowing the key, it should be impossible to recognize the decryption of a modified block, or to predict the effect of a modification on the decrypted block.
- The decryption and encryption operations should be equally secure when used for encryption.

To fulfill the above requirements we use a variable-length block cipher adapted to the length of the payload — that is, a cipher that acts as a permutation on a block the size of its input (a header or the payload). One candidate is LIONESS [2]. The cryptographic property required is that of a super-pseudo-random permutation (a.k.a. strong pseudo-random permutation) or SPRP [20].[2] Thus if any bit of the encrypted material is changed, the decryption will look like random bits. An SPRP is also equally secure in the encryption and decryption directions. In the following section, we describe how this approach helps protect against tagging.

4.2 Defenses against tagging attacks

To motivate the Mixminion design, we describe an attack that works against many mix-net protocols, including Mixmaster and Babel.

A *tagging attack* is an active attack in which a message is modified by altering part of it (for example by flipping bits) so that it can be recognized later in the path. A later mix controlled by the attacker can recognize tagged messages because the header or the body does not conform to the expected format when decrypted. Also, the final recipient can recognize a tagged message for which the payload has been altered. Thus, an attacker can use tagging to trace a message from the point at which it is tagged to the point at which the corrupted output appears.

Checking the integrity of hop headers individually is not sufficient to prevent tagging attacks. For example, in Mixmaster each hop header contains a hash of the other fields in that header [26]. Each mix in the path checks the integrity of its own header and drops the message immediately if it has been altered. However, an attacking mix can still alter a header that will be decrypted only after several more hops, and so tagging attacks are still possible.

The most straightforward way to prevent tagging attacks is to verify the integrity of the whole message at every hop. For forward messages, then, the padding added to a message must be derived deterministically, so that it is possible to calculate authentication tags for the whole message at each hop. But the situation becomes more complicated when reply messages are introduced – the payload and the reply block are created by different users. It is impossible for the creator of the SURB to include in the header a checksum of a message he does not yet know. Therefore conventional techniques such as semantically secure or randomized encryption, that make sure an adversary does not gain any information by sending malformed messages to the mix (since the mix acts as a decryption oracle), cannot be used.

Mixminion uses a hybrid strategy to protect against such attacks: we use cryptographic checksums to protect the headers, and the "swap" step described above insures that the addressing information contained in the headers is destroyed if the payload is modified by an adversary.

If the Mixminion design did not require the crossover point, an adversary could mount a tagging attack by modifying the payload of a forward message as it leaves Alice and recognizing it later when it reaches Bob. For example, if our encryption mechanism were an ordinary counter-mode cipher, an adversary might alter a specific byte in the payload of a message entering the mix-net. Since many of the outgoing messages will be in part predictable (either entirely plaintext, or with predictable PGP header material), the adversary could later observe messages exiting the mix-net and look for payloads that have a corresponding anomaly at that byte. Other cipher modes such as Cipher Block Chaining (CBC) present comparable problems, since whole blocks would look like random bytes instead of the normal payload.

We use a large-block cipher as described in the previous section to minimize the amount of information an adversary

[2] The weaker PRP property may be sufficient, given that preventing replays limits the number of oracle queries to 1; this will need further analysis. In that case the simpler BEAR construction [2] could be used instead of LIONESS.

can learn from tagging. If he tags a message leaving Alice, the payload will be entirely random when it reaches Bob. Thus, an adversary who tags a message can at worst turn the corresponding payload into trash (pseudorandom bit strings entirely unpredictable to the attacker).

We briefly considered introducing *cover-trash*, dummy messages designed to look like tagged messages, to frustrate these tagging attacks; but that problem is as complex as the dummy traffic problem [5]. Instead, we use the "swap" step at the crossover point to prevent the attacker from learning information from tagging attacks. The second header of the message, which contains the path to the final destination of the forward path, is encrypted by the sender with the hash of the payload that is to arrive at the mix. The mix then needs to perform the decryption and swap the first header for the second one. Our security argument has three cases:

- Forward messages: if the message is tagged during the first leg, the second header is unrecoverable, and so the adversary cannot learn the intended destination of the message. If the message is tagged during the second leg, then the first leg has already provided anonymity, and so the adversary cannot learn the sender.
- Direct reply messages: since the decryption algorithm provides secrecy equivalent to encryption, the effect is similar to *encrypting* the payload at each step along a reply block. Only the recipient can learn, after peeling off all layers, whether the message has been tagged. Thus tagging attacks are useless against direct reply messages.
- Anonymized reply messages: as with forward messages, if the first leg is tagged the second header is unrecoverable — so an adversary will never learn that the message was addressed to a reply block. And as with direct reply messages, only the recipient can learn if the second leg is tagged.

While direct reply messages do not need a crossover point in the path (the adversary can never observe his tag), forward messages still need a crossover point to prevent end-to-end tagging. But since the first leg either provides sufficient anonymity or destroys the information about the second leg, the second leg in a forward message can be short. At the extreme, the first hop in the second header could directly specify the message recipient. However, the choice of crossover point can still reveal information about the intended recipient (imagine that some mixes only allow outgoing mail to local addresses; if such a node gets a crossover message that has been trashed, it might guess that the recipient is one of the local addresses), and so we recommend that the second leg be at least a few hops long. We use a path length of 4 hops per leg, but with only 2 hops in the second leg of a forward message.

It is worth noting that while semantically secure encryption cannot be used directly to solve the problem of tagging attacks in Mixminion, the structure of the operations performed on the message as it travels through the network is similar to the Luby-Rackoff [20] structure. In particular the fact that the header depends on the body and *vice versa* makes sure that if the message is tagged in any way it will become entirely different from what was intended, and its contents will provide no information at all to an attacker. Mixminion is the first system, to our knowledge, to achieve this property by distributing the operation of a cipher across many nodes of a mix network.

No mix except the crossover point can get any information distinguishing forward messages from replies. While the crossover point cannot be certain whether the message that it is processing is a forward message or a reply, it does gain partial information because crossover points are less frequent on forward paths, and therefore a message which is crossing-over is more likely to be a reply message.

4.3. Multiple-message tagging attacks

The above design is still vulnerable to a subtle and dangerous attack. If Alice sends a group of messages along the same path, the adversary can tag some of those message as they leave Alice, recognize the pattern (number and timing of tagged and untagged messages) at the crossover point, and observe where the untagged ones go. With some assumptions about our adversary, we can reduce this attack to a traffic confirmation attack we're already willing to accept: when Alice sends a bunch of messages, the adversary can count them and look for the pattern later. He can also drop some of them and look for resulting patterns.

The adversary can only recognize a tag if he happens to own the crossover point that Alice chooses. Therefore, Alice picks k crossover points for her messages;[3] to match a tag signature with certainty, an adversary would have to own all k crossover points. (And even then, it seems harder as the subsets of her messages would overlap with subsets of messages from other senders.)

The key here is that when the adversary doesn't own a given crossover point, tagging messages destined for that crossover is equivalent to dropping them. The crossover point in question simply doesn't deliver the message to the second leg. Therefore, if the adversary doesn't own most of the crossover points that Alice chooses, a successful multiple-message tagging attack seems infeasible. We leave a security analysis of the multiple-paths idea to future work, but see Section 8.

[3] We can prevent the adversary from using divide-and-conquer on Alice's groupings if Alice uses a hybrid path starting with a short cascade — so even if the adversary tags a subset of the messages he doesn't know (unless he owns the whole cascade) the groupings of tagged messages.

5. Other design decisions

5.1. Forward secure link encryption and its benefits

Unlike remailer Types I and II that used SMTP [33] (ordinary Internet e-mail) as their underlying transport mechanism, Mixminion clients and nodes communicate using a forward secure encrypted channel based on TLS [10]. TLS allows the establishment of an encrypted tunnel using ephemeral Diffie-Hellman key negotiation. In order to guarantee that the receiving end is the one intended by the creator of the anonymous message, the receiving node signs the ephemeral key. As soon as a session key has been established, the parties destroy their Diffie-Hellman keys and begin sending messages through the tunnel. After each message, the parties perform a standard key update operation to generate a fresh session key and delete the old key material. Key updates don't require any asymmetric encryption techniques, and so are relatively fast.

The purpose of link encryption is to provide forward secrecy: once the ephemeral link keys have been deleted, not even the nodes that exchange messages can decrypt or recognize messages that might have been intercepted on the links. This makes it impossible to comply with demands for decryption of past traffic that might be served in some jurisdictions, and limits the impact of server compromise on the anonymity of messages already delivered. Even if an attacker manages to get hold of the session key at a particular point he would have to observe all subsequent traffic to be able to update his key appropriately.

Additionally link encryption makes active and passive attacks on the network links more difficult. Since a message tells each mix the identity of its successor in the path, it is difficult for an attacker to mount a man-in-the-middle attack to modify messages, inject messages to a node as if they were part of the normal communications, or delete messages. An additional *heartbeat* signal in the TLS tunnel could be used to complicate message delaying attacks.

The encrypted channel offers only limited protection against traffic analysis. Encrypted links between honest nodes prevent an adversary from recognizing even his own messages, but without link padding, he can still measure how much traffic is being transmitted.

As a fringe benefit, using a separate link protocol makes it easier to deploy relay-only mixes — such nodes simply omit SMTP support, as the next section discusses.

5.2. Message types and delivery modules

Once a Mixminion packet reaches the final mix in its path, it must either be delivered to its intended recipient, dropped if it is an intra-network dummy message, or processed further if it is a remixed Type II packet. In order to support different kinds of delivery, the header includes a type code for the action to be taken to deliver the message. A few types — such as 'dummy', 'SMTP', and 'local delivery' — are specified as a part of the Mixminion standard. Others may be added by future extensions to implement abuse-resistant exit policies (see Section 5.3), to administer nymservers (see Section 7), to publish anonymously to Usenet, to relay messages to older remailers, or to support other protocols.

Nearly all delivery methods require additional information beyond the message type and its payload. The SMTP module, for example, requires a mailbox.[4] This information is placed in a variable-length annex to the final subheader.

The types each mix supports are described in a *capability block*, which also includes the mix's address, long-term (signing) public key, short-term (message decryption) public key, remixing capability, and batching strategy. Mixes sign these capability blocks and publish them on directory servers (see Section 6). Clients download this information from the directory servers.

The possibility of multiple delivery methods doesn't come free: their presence may fragment the anonymity set. For example, if there were five ways to send an SMTP message to Bob, an attacker could partition Bob's incoming mail by guessing that one of those ways is Alice's favorite. An active attacker could even lure users into using a compromised exit node by advertising that node as supporting a rare but desirable delivery method. We believe these attacks do not provide an argument against extensibility *per se*, but rather argue against the proliferation of redundant extensions, and against the use of rare extensions.

5.3. Exit policies and abuse

One important entry in a node's capability block is its *exit policy*, that describes to which addresses and by which methods a mix node is prepared to deliver messages. Exit abuse is a serious barrier to wide-scale remailer deployment — rare indeed is the network administrator tolerant of machines that potentially deliver hate mail.

On one end of the spectrum are *open exit* nodes that will deliver anywhere; on the other end are *middleman* nodes that only relay traffic to other remailer nodes, and *private exit* nodes that only deliver locally. More generally, nodes can set individual exit policies to declare which traffic they will deliver: some may allow traffic only for local users; others may require other forms of traffic authentication [38].

Preventing abuse of open exit nodes is an unsolved problem. If receiving mail is opt-in, an abuser can forge an opt-in request from his victim. Indeed, requiring recipients to

[4] A *mailbox* is the canonical form of the "`user@domain`" part of an e-mail address. Mixminion uses only mailboxes in the protocol, because the other parts of an e-mail address could differ among senders who obtain an address from different sources, thus leading to smaller anonymity sets.

declare their interest in receiving anonymous mail is risky — human rights activists in Guatemala cannot both sign up to receive anonymous mail and also retain plausible deniability. Similarly, if receiving mail is opt-out, an abuser can deny service by forging an opt-out request from a legitimate user. We use a compromise, where all users are assumed to want to receive mail, but each Mixminion message arrives with instructions on how to opt out. Specifically, the message includes a secret that must be used to authorize the opt-out. Thus adversaries who cannot read the victim's mail cannot forge an opt-out request. (We believe that restricting ourselves to such adversaries is reasonable. After all, adversaries strong enough to read the victim's mail can probably deny service to him in some other way. Users may also avoid this attack by running their own 'delivery-only' nodes, which would amount to an implicit opt-in.)

Of course, a mixture of open and restricted exit nodes will allow the most flexibility for volunteers running servers. But while a large number of middleman nodes is useful to provide a large and robust network, a small number of exit nodes still simplifies traffic analysis. In these attacks, the adversary observes both a suspected user and the network's exit nodes and looks for timing or packet correlations. The fewer exit nodes in the system, the easier it is for an attacker to observe them all. Thus, the number of available open exit nodes remains a limiting security parameter for the remailer network.

5.4. Replay prevention, message expiration, and key rotation

Mixmaster offers rudimentary replay prevention by keeping a list of recent message IDs. To keep the list from getting too large, it expires entries after a server-configurable amount of time. But if an adversary records the input and output batches of a mix and then replays a message after the mix has forgotten about it, the message's decryption will be exactly the same. Thus, Mixmaster does not provide the forward anonymity that we want.

Chaum first observed this attack in [7], but his solution (which is proposed again in Babel[5]) — to include in each message a timestamp that describes when that message is valid — also has problems. Specifically, it introduces a new class of partitioning attacks, where the adversary can distinguish and track messages based on timestamps. If messages have short lifetimes, then some legitimate messages will expire before they can be delivered. But if messages have long lifetimes, then messages near their expiration date will be rare, and an adversary can exploit this fact by intentionally delaying a message until near its expiration date. If he owns a mix later in the path he can recognize the message by its unusually late expiration date.

One way of addressing this partitioning attack is to add dummy traffic so that it is less rare for messages to arrive near their expiration date, but dummy traffic is still not well-understood. Another approach would be to add random values to the expiration date of each mix in the path, so an adversary delaying a message at one mix cannot expect that it is now near to expiring elsewhere in the path; but this seems open to statistical attacks.

We use a compromise solution that still provides forward anonymity. Messages don't contain any timestamp or expiration information. As in Mixmaster, each mix keeps hashes of the headers of all messages it has processed; but unlike Mixmaster, a mix only discards these hashes when it rotates its public key. Mixes should choose key rotation frequency based on their security goals and on the number of hashes they are willing to store, and advertise their key rotation schedules along with their public key information. (See Section 6.)

Note that this solution does not entirely solve the partitioning problem — near the time of a key rotation, the anonymity set of messages will be divided into those senders who knew about the key rotation and used the new key, and those who did not. Also note that while key rotation and link encryption (see Section 5.1) both provide forward security, their protection is not redundant. With only link encryption, an adversary running one mix could compromise another and use its private key to decrypt messages previously sent between them. Key rotation limits the window of opportunity for this attack.

6. Directory Servers

The Mixmaster protocol does not specify a means for clients to learn the locations, keys, capabilities, or performance statistics of mixes. Several *ad hoc* schemes have grown to fill that void [30], but as we explain below, it is important that all clients learn this information in the same way. (Omitting directory servers is not an option: without timely information, clients cannot respond to changes in the set of mixes, or to changes in mix keys.) Here we describe Mixminion directory servers and examine the anonymity risks of such information services.

In Mixminion, a group of redundant directory servers provide clients information about nodes' current keys, capabilities, and state. These directory servers must be synchronized and redundant: we lose security if clients have different information about network topology and node reliability. An adversary who controls a directory server could track certain clients by providing different information —

[5]Actually, Babel is vulnerable to a much more direct timestamp attack: each layer of the onion includes "the number of seconds elapsed since January 1, 1970 GMT, to the moment of message composition by the sender." Few people will be composing a message on a given second, so an adversary owning a mix at the beginning of the path and another at the end could trivially recognize a message.

perhaps by listing only mixes under its control, or by informing only certain clients about a given mix.

Moreover, an adversary without control of a directory server can still exploit differences among client knowledge. If Eve knows that mix M is listed on server D_1 but not on D_2, she can use this knowledge to link traffic through M to clients who have queried D_1. Eve can also distinguish traffic based on any differences between clients who use directory servers and those who don't, between clients with up-to-date listings and those with old listings, and (if the directory is large and so is given out in pieces) between clients who have different subsets of the directory.

So it is not merely a matter of convenience for clients to retrieve up-to-date mix information. We must specify a directory service as a part of our standard. Thus Mixminion provides protocols for mixes to advertise their capability certificates to directory servers, and for clients to download *complete* directories.[6] Directory servers work together to ensure correct and complete data by successively signing certificate bundles, so users can be sure that a given mix certificate has been seen by a threshold of directory servers. While we require stronger synchronization and trust for the directory servers, we believe this is realistic because there will be far fewer of them than mix nodes, and they will be much more static.

But even if client knowledge is uniform, an attacker can mount a *trickle attack* by delaying messages from Alice at a compromised node until the directory servers remove some mix M from their listings — he can then release the delayed messages and guess that any messages still using M are likely to be from Alice. An adversary controlling many nodes can launch this attack effectively. Thus clients should download new information regularly, but wait for a given time threshold (say, an hour) before using any newly-published nodes. Dummy traffic to old nodes may also help thwart trickle attacks.

Directory servers compile node availability and performance information by sending traffic through mixes in their directories. This is currently similar to the current ping servers [30], but in the future we can investigate integrating more complex and attack-resistant reputation metrics. But even this reputation information introduces vulnerabilities: for example, an adversary trying to do traffic analysis can get more traffic by gaining a high reputation [11]. We can defend against these attacks by building paths from a suitably large pool of nodes [12] to bound the probability that an adversary will control an entire path, but there will always be a tension between giving clients accurate and timely information and preventing adversaries from exploiting the directory servers to manipulate client behavior.

[6] We recommend against retrieving anything less than a complete directory. Even if clients use the mix-net to anonymously retrieve a random subset of the directory, an adversary observing the directory servers and given two hops in a message's path can take the intersection over recently downloaded directory subsets to guess the remaining hops in the path. Private Information Retrieval [21] may down the road allow clients to efficiently, securely, and privately download a subset of the directory.

7. Nym management and single-use reply blocks

Current nymservers, such as `nym.alias.net` [22], maintain a set of $\langle$mailbox, reply block$\rangle$ pairs to allow users to receive mail without revealing their identities. When mail arrives to `<bob@nym.alias.net>`, the nymserver attaches the payload to the associated reply block and sends it off into the mix-net. Because these nymservers use the Type I remailer network, these reply blocks are *persistent* or *long-lived* nyms — the mix network does not drop replayed messages, so the reply blocks can be used again and again. Reply block management is much simpler in this model because users only need to replace a reply block when one of the nodes it uses stops working.

The Mixminion design protects against replay attacks by dropping messages with repeated headers — so its reply blocks are necessarily single-use. Nonetheless, there are still a number of approaches for building nymservers from single-use reply blocks.

In the first approach, nymservers keep a stock of reply blocks for each mailbox, and use a new reply block α_i for each incoming message. Suppose Alice wants to register a pseudonym α with signature and verification keys (S_α, V_α) with the Nym server in order to receive messages from Bob. In this case, the parties communicate as follows:

$$\begin{aligned} &\alpha \rightarrow \mathrm{Nym} : \{\mathrm{Register}, \alpha, V_\alpha, \alpha_1 \ldots \alpha_n\}_{S_\alpha} \\ &B \rightarrow \mathrm{Nym} : \alpha, M \\ &\mathrm{Nym} \rightarrow \alpha_i : M \end{aligned} \tag{1}$$

As long as the owner of the pseudonym keeps the nymserver well-stocked, no messages will be lost. But it is hard for the user to know how many new reply blocks to send; indeed, under this approach, an attacker can deny service by flooding the mailbox to exhaust the available reply blocks and block further messages from getting delivered.

A more robust design uses a protocol inspired by e-mail retrieval protocols such as POP [27]: messages arrive and queue at the nymserver, and the user periodically checks the status of his mail and sends a sufficient batch of reply blocks so the nymserver can deliver that mail. In this case, the parties communicate as follows:

$$\begin{aligned} &\alpha \rightarrow \mathrm{Nym} : \{\mathrm{Register}, \alpha, V_\alpha\}_{S_\alpha} \\ &B \rightarrow \mathrm{Nym} : \alpha, M \\ &\alpha \rightarrow \mathrm{Nym} : \{\mathrm{Query}, \alpha, \alpha_1 \ldots \alpha_n\}_{S_\alpha} \\ &\mathrm{Nym} \rightarrow \alpha_i : M \end{aligned} \tag{2}$$

In this case, the nymserver doesn't need to store any reply blocks. The above flooding attack still works, but now it is exactly like flooding a normal POP mailbox, and the usual techniques (such as allowing the user to delete mails at the server or specify which mails to download and let the others expire) work fine. The user can send a set of indices to the server after successfully receiving some messages, to indicate that they can now be deleted.

Of course, there are different legal and security implications for the two designs. In the first design, no mail is stored on the server, but it must keep valid reply blocks on hand. The second case is in some sense more secure because the server need not store any reply blocks, but it also creates more liability because the server keeps mail for each recipient until it is retrieved. The owner of the pseudonym could provide a public key that the nymserver uses to immediately encrypt all incoming messages to limit the amount of time the nymserver keeps plaintext messages.

The best implementation depends on the situations and preferences of the volunteers running the nymservers. We hope that as we gain more experience with their needs and the needs of their users, we will converge on a suitable model.

8. Maintaining anonymity sets

8.1. Batching Strategy

Low-latency systems like Onion Routing aim to provide anonymity against an adversary who is not watching both Alice and Bob [39]. If the adversary watches both, he can for instance count packets and observe packet timing to become confident that they are communicating. Because Mixminion aims to defeat even a global passive adversary, we must address this end-to-end timing vulnerability.

Further, because our adversary can send and delay messages, he can manipulate the batch of messages entering a mix so the only message unknown to him in the batch is the target message. This approach is known as the *blending attack* because the adversary blends his own recognizable messages with the honest messages in the batch [36]. By repeatedly attacking each mix in the path, the adversary will link Alice and Bob.

Mixminion nodes use a *timed dynamic-pool* batching strategy [36] adapted from Mixmaster. Rather than simply processing each message as soon as it arrives, each mix keeps a pool of messages. New messages arrive, are decrypted, and enter the pool. The mix fires every t seconds, but only if the pool contains at least a threshold of messages. If the mix fires, it randomly chooses a constant fraction of the pool messages (say, 60%) to deliver.

Since the number of messages delivered each round is based on the rate of incoming messages, an attacker cannot overflow the pool with sustained flooding. These mixes also increase the cost of the blending attack: while the *number* of messages coming out increases as the rate of incoming messages increases, the chance that any given message will leave the pool remains constant. Thus it is impossible to arrange to completely flush the mix with high probability in one flush. An adversary is forced to spend multiple intervals (and thus delay other messages for considerable time) first to flush the original honest messages from the mix, and again to flush the target message from the mix. This delay can be noticed by the other mixes, because they communicate over TLS with a heartbeat to detect delays.

This batching strategy also increases the cost of intersection attacks by providing large anonymity sets for each message in the network. Because a message could plausibly have been held in a pool for several rounds at each mix, the set of possible senders when Bob receives the target message is large.

8.2. Dummy policy

Dummy traffic (sending extra messages that are not actually meant to be read or used, to confuse the adversary) is an old approach to improving anonymity, but its efficacy is still not well analyzed.

One use for dummies is to weaken the intersection attack, perhaps by letting mixes introduce dummies addressed to actual users. But to do this, each mix must know all the users in the system: if a mix only delivers dummies to a subset of the users, an adversary can distinguish with better than even probability between a dummy and a legitimate message. While there is some initial research on the subject [5], we currently know no practical way to use dummies to provably help against the intersection attack. Thus Mixminion does not at present incorporate dummies to or from users.

Instead, we incorporate mix-to-mix dummies to weaken the blending attack. As described in Section 8.1 above, our timed dynamic-pool batching strategy already increases the cost of the blending attack because the adversary needs to keep flushing the mix until all honest messages are out — but once the adversary has done so, he can be certain that no honest messages remain. In the second phase of the attack, he again needs to flush until the target message comes out, but once it does, he can be certain of recognizing it. To prevent this, Mixminion employs the following dummy policy, as suggested in [36]: each time the mix fires, it also sends out a number of dummies chosen from a geometric distribution. These dummies travel a number of hops chosen uniformly between 1 and 4. The blending attack is now harder — the adversary can no longer single out the target message in the outgoing batch, and so must track each of the dummies along with the original target message.

During normal traffic, these dummies have little effect on anonymity. They aim to protect anonymity in times of low traffic — either when there are actually few messages going through the mix, or when most messages are created by the adversary.

8.3. Choosing paths when transmitting many messages

When Alice (the owner of a pseudonym) downloads her mail from a nymserver, she will likely receive many separate messages. Similarly, if Alice uses Mixminion as a transport layer for higher-level applications, sending a large file means sending many Mixminion messages, because of their fixed size. Conventional wisdom suggests that she should pick a different path for every message, but an adversary that owns all the nodes in *any* of the paths could learn her identity — without any work at all. Even an adversary owning a small fraction of the network can perform this attack, since each Mixminion payload is small.

Alice thus seems most likely to maintain her unlinkability by sending all the messages over the same path. On the other hand, a passive adversary can watch the flood of messages traverse that path.

A compromise approach is to pick a small number of paths and use them together. By sending out the messages over time rather than all at once, and assuming more people than just Bob are receiving many messages, the pool mixes will create a large anonymity set of possible senders. However, a complete solution to the intersection attack remains an open problem.

9. Attacks and Defenses

Below we summarize a variety of attacks and how well our design withstands them.

1. **Mix attacks**

    - *Compromise a mix.* Messages traverse multiple mixes, so compromising a single mix, even a crossover point, does not gain much.
    - *Compromise a mix's private key.* Again, controlling a single mix is of limited use. Further, periodic mix key rotation limits the window of time in which to attack the next mix in the target message's path.
    - *Replaying messages.* Mixes remember header cryptographic checksums of previously seen messages; after key rotation these old headers can no longer be decrypted.
    - *Delaying messages.* The adversary can delay messages and release them when certain network parameters (eg traffic volume) are different. The efficacy of this attack is poorly understood, but it may well be quite damaging [36]. Imposing a deadline on transmission at each hop may help [11].
    - *Dropping messages.* The adversary can drop messages with the hope that users will notice and resend. If the user must resend, she should use the same path, to prevent the adversary from forcing her onto an adversary-controlled path (see Section 8.3).
    - *Tagging messages.* Mixes detect modified headers immediately using checksums. The payload can still be tagged, but the "swap" step along with SPRP encryption from Section 4.1 provide protection.
    - *$N-1$ attack (trickle, flooding)* The "timed dynamic-pool" batching strategy from Section 8.1, along with our dummy policy, limits the effectiveness of these blending attacks.

2. **Passive attacks**

    - *Intersection attack.* Our dynamic-pool batching strategy from Section 8.1 spreads out the messages over time, increasing the set of possible senders for a given received message and thus increasing the cost of an intersection attack. However, a complete solution remains an open problem [5].
    - *Textual analysis.* Mixminion provides location anonymity, not data anonymity. Users are responsible for making sure their messages do not reveal identifying information. Such attacks are practical, and therefore a real threat, as documented in [34].

3. **Exit attacks**

    - *Partition traffic by delivery method.* We encourage recipients to use one of only a few delivery methods, so we can maintain sufficient anonymity sets for each.
    - *Partition traffic by exit capabilities.* Delivery methods should be standardized; users should be suspicious of delivery methods only offered by a few exit nodes.
    - *Use the mix network to send hate mail, etc.* We allow recipients to opt out of receiving further mail. Still, we must have enough nodes that

can withstand complaints stemming from abusive email, or it will be too easy for an adversary to monitor all exit nodes in the network.

4. **Directory attacks**

   - *Compromise a directory server.* Identical directory listings are served by a small group of servers and signed by all. We assume that a threshold of these directory servers will remain honest.
   - *Exploit differences in client directory knowledge.* By only updating directory information nightly, by designing client software to pull updates as soon as possible after their release, and by ensuring that clients have the entire directory, we can limit this attack.
   - *Delay mix messages until directory information changes.* The fact that clients delay using new information, along with dummy traffic sent to delisted destinations and expired keys, should mitigate this attack. Again, imposing a deadline on transmission at each hop may help more [11].
   - *Sign somebody else up as a mix.* Signatures on capability blocks prevent others from forging blocks to the directory servers.
   - *Flood the directories with nonfunctional mix entries; run highly reliable mixes to gain traffic for analysis; attack honest mixes to encourage users to start using the dishonest ones.* Availability and reliability statistics should mitigate some of these problems, but they introduce problems of their own. They are an area of active research [11, 12].

10. Future Directions and Open Problems

This design document represents the first step in peer review of the Type III remailer protocol. Many of the ideas, ranging from the core design to peripheral design choices, need more attention:

- We need more research on batching strategies that resist blending attacks [36] as well as intersection attacks on asynchronous free routes [6]. In particular the anonymity they provide during normal operation or under attack must be balanced with other properties such as latency and reliability.

- We need a more thorough investigation of multiple-message tagging attacks and an analysis of how to safely choose paths when sending many messages. When a message to be sent is larger than the Mixminion payload size, we need a strategy to fragment it and reconstruct it at the recipient's end. We can use retransmission strategies or forward error correction codes to recover if some messages are lost.

- Can we keep the indistinguishability of forward messages and replies using a simpler design? We need to prove that our design provides *bit-wise unlinkability* between the input bit-patterns of messages and the messages coming out of the network.

- Currently, reply messages can be distinguished from plaintext forward messages at the exit nodes: the former exit as encrypted data, and the latter do not. We prevent further partitioning by arranging encrypted forward messages to blend in with the reply messages, but even this degree of distinguishability is unsettling. Finding further means to mitigate this problem would be helpful.

- A *synchronous batching* approach, where messages have deadlines for each hop, may allow easier anonymity analysis, and may provide much larger anonymity sets because all messages entering the mix-net in a given time interval are mixed together. A cascade is the simplest example of this approach, but we should consider mechanisms for free-route synchronous mixes. We could greatly improve our protection against message delaying attacks and the partitioning attacks discussed in Section 5.4. On the other hand, the costs are greater network synchronization and overhead, and less mix operator flexibility.

- We need stronger intuition about how to use dummy messages. Such messages can be inserted between nodes as link padding, or as actual multi-hop Mixminion messages. We must develop a more analytically justified approach to determine which parties send dummy messages, how many they send, and when they send them.

  While many people have speculated about the benefits of dummy traffic, we have not yet seen any convincing analysis. For this reason, while Mixminion is flexible enough to support them, we plan to leave dummies out of the design (other than their minimal use in Section 8.1) until their effects on anonymity are better understood.

We have working code which implements most of the designs described in this paper, with acceptable performance even using 2048 bit RSA keys (800KB of messages per second on a 1GHz Athlon). We invite interested developers to join the `mixminion-dev` mailing list and examine the more detailed Mixminion specification [24].

11. Acknowledgments

This paper incorporates ideas from the Mixmaster development team, particularly Len Sassaman, Scott Renfro, Peter Palfrader, Ulf Möller, Lance Cottrell, and Bram Cohen, to improve the Type II remailer protocol; their effort was abandoned in favor of Mixminion.

Susan Born, Lucky Green, David Hopwood, David Mazières, Peter Palfrader, Len Sassaman, Andrei Serjantov, Robyn Wagner, and Bryce "Zooko" Wilcox-O'Hearn provided helpful design discussions, editing, and suggestions. We further thank all the unnamed cypherpunks out there who have worked on remailer issues for the past decades.

References

[1] M. Abe. Universally verifiable MIX with verification work independent of the number of MIX servers. In *EUROCRYPT 1998*. Springer-Verlag, LNCS 1403, 1998.

[2] R. Anderson and E. Biham. Two practical and provably secure block ciphers: BEAR and LION. In *International Workshop on Fast Software Encryption*. Springer-Verlag, 1996. <http://citeseer.nj.nec.com/anderson96two.html>.

[3] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001. <http://www.cypherspace.org/adam/pubs/traffic.pdf>.

[4] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, 2000.

[5] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In R. Dingledine and P. Syverson, editors, *Privacy Enhancing Technologies (PET 2002)*. Springer-Verlag, LNCS 2482, 2002.

[6] O. Berthold, A. Pfitzmann, and R. Standtke. The disadvantages of free MIX routes and how to overcome them. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 30–45. Springer-Verlag, LNCS 2009, 2000. <http://www.tik.ee.ethz.ch/~weiler/lehre/netsec/Unterlagen/anon/disadvantages_berthold.pdf>.

[7] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudo-nyms. *Communications of the ACM*, 4(2), February 1982. <http://www.eskimo.com/~weidai/mix-net.txt>.

[8] L. Cottrell. Mixmaster and remailer attacks. <http://www.obscura.com/~loki/remailer/remailer-essay.html>.

[9] Y. Desmedt and K. Kurosawa. How to break a practical MIX and design a new one. In *EUROCRYPT 2000*. Springer-Verlag, LNCS 1803, 2000. <http://citeseer.nj.nec.com/447709.html>.

[10] T. Dierks and C. Allen. The TLS Protocol — Version 1.0. IETF RFC 2246, January 1999. <http://www.rfc-editor.org/rfc/rfc2246.txt>.

[11] R. Dingledine, M. J. Freedman, D. Hopwood, and D. Molnar. A Reputation System to Increase MIX-net Reliability. In I. S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 126–141. Springer-Verlag, LNCS 2137, 2001. <http://www.freehaven.net/papers.html>.

[12] R. Dingledine and P. Syverson. Reliable MIX Cascade Networks through Reputation. In M. Blaze, editor, *Financial Cryptography (FC '02)*. Springer-Verlag, LNCS (forthcoming), 2002. <http://www.freehaven.net/papers.html>.

[13] J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *CRYPTO 2001*. Springer-Verlag, LNCS 2139, 2001.

[14] C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium - NDSS '96*. IEEE, 1996. <http://citeseer.nj.nec.com/2254.html>.

[15] J. Helsingius. anon.penet.fi press release. <http://www.penet.fi/press-english.html>.

[16] M. Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM Press, 1999. <http://citeseer.nj.nec.com/jakobsson99flash.html>.

[17] M. Jakobsson and A. Juels. An optimally robust hybrid mix network (extended abstract). In *Principles of Distributed Computing - PODC '01*. ACM Press, 2001. <http://citeseer.nj.nec.com/492015.html>.

[18] A. Jerichow, J. Müller, A. Pfitzmann, B. Pfitzmann, and M. Waidner. Real-Time MIXes: A Bandwidth-Efficient Anonymity Protocol. IEEE Journal on Selected Areas in Communications, 1998. <http://www.zurich.ibm.com/security/publications/1998.html>.

[19] D. Kesdogan, M. Egner, and T. Büschkes. Stop-and-go MIXes providing probabilistic anonymity in an open system. In *Information Hiding (IH 1998)*. Springer-Verlag, LNCS 1525, 1998. <http://www.cl.cam.ac.uk/~fapp2/ihw98/ihw98-sgmix.pdf>.

[20] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[21] T. Malkin. *Private Information Retrieval*. PhD thesis, MIT, 2000. <http://toc.lcs.mit.edu/~tal/pubs.html>.

[22] D. Mazières and M. F. Kaashoek. The Design, Implementation and Operation of an Email Pseudonym Server. In 5^{th} *ACM Conference on Computer and Communications Security (CCS'98)*. ACM Press, 1998. <http://www.scs.cs.nyu.edu/~dm/>.

[23] M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *ASIACRYPT 2000*. Springer-Verlag, LNCS 1976, 2000. <http://citeseer.nj.nec.com/450148.html>.

[24] Mixminion. Type III (Mixminion) mix protocol specifications.
<http://mixminion.net/minion-spec.txt>.

[25] B. Möller. Provably secure public-key encryption for length-preserving chaumian mixes. In *CT-RSA 2003*. Springer-Verlag, LNCS 2612, 2003.

[26] U. Möller and L. Cottrell. Mixmaster Protocol — Version 2. Unfinished draft, January 2000. <http://www.eskimo.com/~rowdenw/crypt/Mix/draft-moeller-mixmaster2-protocol-00.txt>.

[27] J. Myers and M. Rose. Post Office Protocol — Version 3. IETF RFC 1939 (also STD0053), May 1996. <http://www.rfc-editor.org/rfc/rfc1939.txt>.

[28] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In P. Samarati, editor, *8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 116–125. ACM Press, November 2001. <http://www.votehere.net/ada_compliant/ourtechnology/technicaldocs/shuffle.pdf>.

[29] M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000*. Springer-Verlag, LNCS 1976, 2000.

[30] P. Palfrader. Echolot: a pinger for anonymous remailers. <http://www.palfrader.org/echolot/>.

[31] S. Parekh. Prospects for remailers. *First Monday*, 1(2), August 1996. <http://www.firstmonday.dk/issues/issue2/remailers/>.

[32] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of MIXes. In *Eurocrypt 89*. Springer-Verlag, LNCS 434, 1990. <http://citeseer.nj.nec.com/pfitzmann90how.html>.

[33] J. Postel. Simple Mail Transfer Protocol. IETF RFC 2821 (also STD0010), April 2001. <http://www.rfc-editor.org/rfc/rfc2821.txt>.

[34] J. R. Rao and P. Rohatgi. Can pseudonymity really guarantee privacy? In *Proceedings of the Ninth USENIX Security Symposium*, pages 85–96. USENIX, Aug. 2000. <http://www.usenix.org/publications/library/proceedings/sec2000/full_papers/rao/rao.pdf>.

[35] J. F. Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 10–29. Springer-Verlag, LNCS 2009, July 2000.

[36] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In F. Petitcolas, editor, *Information Hiding (IH 2002)*. Springer-Verlag, LNCS (forthcoming), 2002.

[37] Z. K. Systems. Freedom version 2 white papers. <http://www.freedom.net/info/whitepapers/>.

[38] P. Syverson, M. Reed, and D. Goldschlag. Onion Routing access configurations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, volume 1, pages 34–40. IEEE CS Press, 2000. <http://www.onion-router.net/Publications.html>.

[39] P. F. Syverson, G. Tsudik, M. G. Reed, and C. E. Landwehr. Towards an analysis of onion routing security. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000. <http://citeseer.nj.nec.com/syverson00towards.html>.