Mixminion: Design of a Type III Anonymous Remailer Protocol

George Danezis[1], Roger Dingledine[2], David Hopwood[3], and Nick Mathewson[2]

[1] Cambridge University <george.danezis@cl.cam.ac.uk>
[2] The Free Haven Project <{arma,nickm}@freehaven.net>
[3] Independent consultant <david.hopwood@zetnet.co.uk>

Abstract. We present Mixminion, a message-based anonymous remailer protocol that supports 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 allow users to maintain long-term pseudonyms using single-use reply blocks as a primitive. Our design integrates link encryption between remailers to provide forward anonymity. Mixminion brings together the best solutions from previous work to create a conservative design that protects against most known attacks.

Keywords: anonymity, MIX-net, peer-to-peer, remailer, nymserver, reply block

1 Introduction

Chaum first introduced anonymous remailer designs over 20 years ago [9]. The research community has since introduced many new designs and proofs [1, 5, 19, 21–23, 34, 35], and discovered a variety of new attacks [4, 6, 7, 12, 29, 38], but the state of deployed remailers has changed remarkably little since Cottrell published his Mixmaster software [10, 32] in 1994. Part of the difficulty in expanding the deployed remailer base is due to the liability involved in running a remailer node on the Internet, and part is due to the complexity of the current infrastructure — it is fairly hard to add new experimental features to the current software.

The Mixminion Project aims to deploy a cleaner remailer design in the same spirit as Mixmaster, with the goals of expanding deployment, documenting our design decisions and how well they stand up to all known attacks, and providing a research base for experimental features. We describe our overall design in Section 3, including a new primitive called a *single-use reply block* (SURB). Mixmaster provides no support for replies, but instead relies on the older and less secure Cypherpunk Type I remailer design [26]. By integrating reply capabilities into Mixminion, we can finally retire the Type I remailer network.

We introduce link encryption with ephemeral keys to ensure forward anonymity for each message. We also provide flexible delivery schemes — rather than

just allowing delivery to mail or Usenet, we allow designers to add arbitrary modules to handle incoming and outgoing messages. By separating the core mixing architecture from these higher-level modules, we can both limit their influence on the anonymity properties of the system, and also extend the Mixminion network for uses other than anonymous email. We go on in Section 5 to describe a design for directory servers to track and distribute remailer availability, performance, and key information, and then describe in Section 6 how to securely build higher-level systems such as nymservers using SURBs.

Mixminion is a best-of-breed remailer protocol that uses conservative design approaches to provide security against most known attacks. The overall Mixminion Project is a joint effort between cryptography and anonymity researchers and Mixmaster remailer operators. This design document represents the first step in peer review of the Type III remailer protocol.

2 Related Work

2.1 MIX-nets

Chaum introduced the concept of a MIX-net for anonymous communications [9]. A MIX-net consists of a group of servers, called MIXes (or MIX nodes), each of which is associated with a public key. Each MIX receives encrypted messages, which are then decrypted, batched, reordered, and forwarded on without any information identifying the sender. Chaum also proved the security of MIXes against a *passive adversary* who can eavesdrop on all communications between MIXes but is unable to observe the reordering inside each MIX.

Recent research on MIX-nets includes stop-and-go MIX-nets [23], distributed flash MIXes [21] and their weaknesses [12, 29], and hybrid MIXes [35].

One type of MIX hierarchy is a cascade. In a cascade network, users choose from a set of fixed paths through the MIX-net. Cascades can provide greater anonymity against a large adversary: free-route systems allow an adversary who owns many MIXes to use intersection attacks to reduce the set of possible senders or receivers for a given message [7]. On the other hand, cascades are more vulnerable [3] to trickle attacks, where an attacker targeting a specific message going into a MIX can manipulate the batch of messages entering that MIX so the only unknown message in the batch is the target message [10, 19]. MIX cascade research includes real-time MIXes [22] and web MIXes [5].

2.2 Deployed Remailer Systems

The first widespread public implementations of MIXes were produced by the Cypherpunks mailing list. These "Type I" *anonymous remailers* were inspired both by the problems surrounding the `anon.penet.fi` service [20], and by theoretical work on MIXes. Hughes wrote the first Cypherpunk anonymous remailer [26]; 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 [17, 32], or "Type II" remailers, which added message padding, message pools, and other MIX features lacking in the Cypherpunk

remailers. Note that Mixmaster does not include replies, so deployed remailer systems still use the less secure long-term Cypherpunk reply blocks.

At about the same time, Gulcu and Tsudik introduced the Babel system [19], a practical remailer design with many desirable features. While it provides replies, they are only indistinguishable from forward messages by passive observers; the MIX nodes can still distinguish. Babel's reply addresses are multiple-use, making them less secure than forward messages due to replay vulnerabilities. Babel also introduces *inter-MIX detours*, where nodes can rewrap a message and send it through a few randomly chosen new hops — so even the sender cannot be sure of recognizing his message as it leaves the MIX.

2.3 Remailer Statistics

Levien's *statistics pages* [27] track both remailer capabilities (such as what kinds of encryption the remailer supports) and remailer up-times (obtained by pinging the machines in question and by sending test messages through each machine or group of machines). Such *reputation systems* improve the reliability of MIX-nets by allowing users to avoid choosing unreliable MIXes. The Jack B Nymble 2 remailer client [39] and the Mixmaster 2.9 remailer allow users to import statistics files and can then pick remailers according to that data. Users can specify minimum reliability scores, decide that a remailer should always or never be used, and specify maximum latency. Ongoing research on more powerful reputation systems includes a reputation system for free-route networks [14] and another for MIX cascades [16].

3 The MIX-net Design

Mixminion brings together the current best approaches for providing anonymity in a batching message-based MIX environment. We don't aim to provide low-latency connection-oriented services like Freedom [40] or Onion Routing [18] — while those designs are more effective for common activities like anonymous web browsing, the low latency necessarily implies smaller anonymity sets than for 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 divide the anonymity set.

Mixminion uses the same general MIX-net paradigm as previous work [9, 10, 19]. The sender Alice chooses a path through the network. She repeatedly "onion" encrypts her message, starting with the last MIX in her path, and sends the onion to the first MIX. Each MIX unwraps a single layer of the onion, pads the message to a fixed length (32 Kbytes in our current design), and passes the result to the next MIX. We describe the behavior of the last MIX in Section 4.2.

Headers addressed to each intermediate MIX are encrypted using RSA. They contain a secret that can be used to generate padding and decrypt the rest of the message. They also contain the address of the next node to which the message should be forwarded along with its expected signature key fingerprint.

While Mixminion protects against known *traffic analysis* attacks (where an adversary attempts to learn a given message's sender or receiver [37, 38]), we do not fully address *traffic confirmation* attacks. In a traffic confirmation attack, the adversary treats the MIX network as a black box and observes the behavior of senders and receivers. Over time, he can intersect the set of senders and receivers who are active at certain times and learn who is sending and receiving which messages [6]. Good dummy traffic designs may eventually address the intersection attack, but for now it remains an open problem.

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 messages to be Type III messages, thus increasing anonymity sets in the Type III network. Type II messages travelling between Type III remailers are treated as plaintext and encrypted to the next remailer in the chain using its Type III key. The message is sent as a Type III encrypted message, but it decrypts to reveal the Type II message.

We also provide a new feature: a reply block mechanism that is as secure as forward messages. Reusable reply blocks, such as those in the Cypherpunk remailer, are a security risk — by their very nature they let people send multiple messages through them. These multiple messages can easily be used 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 therefore provides only *single-use* reply blocks. Since replies may be very 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.

3.1 Tagging attacks

To motivate some aspects of 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 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.

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 [32]. Each MIX in the path checks the integrity of the 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 authenticate 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 message and the reply block are created by different users.

3.2 Replies

The rest of this section describes the mechanism for secure replies, including how we defeat tagging-related attacks. Mixminion's reply model is in part inspired by Babel [19], as it requires the receiver of a reply block to keep no other state than its secret keys, in order to read the reply. All the secrets used to strip the layers of encryption are derived from a master secret contained in the last header of the single-use reply block, which the creator of the block addresses to itself and encrypts under its own public key.

3.3 Indistinguishable replies

By making forward messages and replies indistinguishable even to MIXes, we prevent an adversary from dividing the message anonymity sets into two classes. In particular, if replies are infrequent relative to forward messages, an adversary who controls some of the MIXes can more easily trace the path of each reply.

Having indistinguishable replies, however, makes it more difficult to prevent tagging attacks. Since the author of a reply block is not the one writing the payload, a hash of the entire message cannot be used. Therefore, since we choose to make forward messages and replies indistinguishable, we cannot include hashes for forward messages either. Our approach to defending against these attacks is discussed in more detail in Section 3.4.

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.

We require parties that benefit from anonymity properties to run dedicated software. Specifically, senders generating forward messages must be able to create onions, and anonymous receivers must be able to create reply blocks and unwrap messages received through those reply blocks. Other parties, such as those receiving forward messages and those sending direct reply messages, do not need to run new software. (The quoting performed by ordinary mail software can be used to include the reply block in a direct reply; this is sent to a node at the Reply-To: address, which extracts the reply block and constructs a properly formatted onion.)

Messages are composed of a header section and a payload. We divide a message's path into two *legs*, and split 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 the appropriate MIX, so we can do integrity-checking of the path (but not the payload) within each leg. Each subheader also contains a symmetric key, which is used to derive a decryption key for decrypting the rest of the message. The MIX also derives a padding seed from this master key. It uses this padding seed to place predictable padding at the end of the header, so the hash will match even though each hop must regrow the header to maintain constant length.

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 first leg, or send the reply block and message to a MIX that can wrap them for her.

When Alice creates her message, she encrypts the secondary header with a hash of her payload (in addition to the usual layered onion encryptions). Alice's message then 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 1 — 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. The encryption primitive, labelled "LBC", that is used to blind the second header and the payload needs to have certain properties:

- it is length-preserving;
- it should be impossible to recognize the decryption of a modified block, without knowledge of the key;
- it should be equally secure to use the decryption operation for encryption.

To fulfill the above requirements we use a large-block cipher; that is, a cipher that acts as a permutation on a block the size of its input (a header or the payload). Possible candidates include LIONESS [2] and SPC [8]. The cryptographic property required is that of a super-pseudo-random permutation (a.k.a. strong pseudo-random permutation) or SPRP [24].[1] 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. See Section 3.4 for a discussion of how this approach helps protect against tagging.

3.4 Defenses against tagging attacks

Without 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. Specifically, if our encryption mechanism were an

[1] 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.

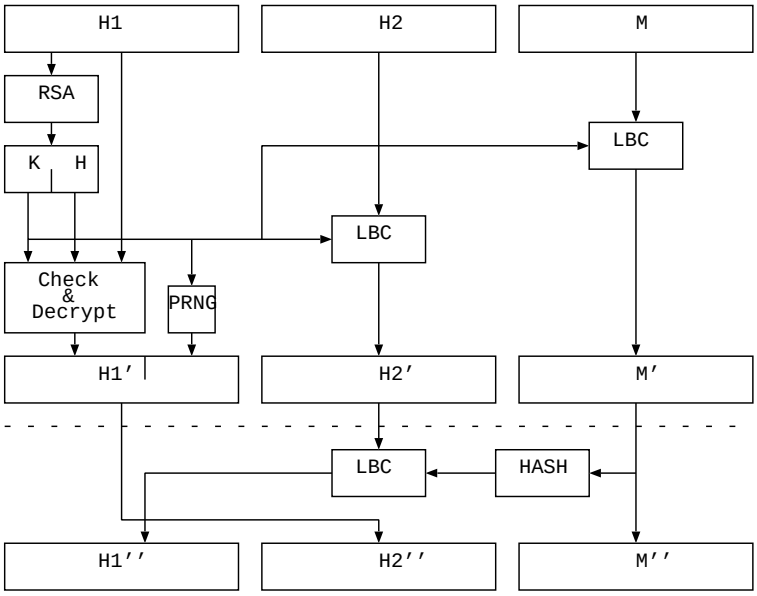

Fig. 1. The operations required by the "swap" method

ordinary counter-mode cipher, he 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 can later observe messages exiting the MIX-net and look for payloads that have a corresponding anomaly at that byte.

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

We briefly considered introducing *cover-trash* to frustrate these tagging attacks; but that problem is as complex as the dummy traffic problem [6]. Instead, we use the decryption-by-hash-of-payload step at the crossover point to prevent the attacker from learning information from tagging attacks. Specifically, our solution falls into several 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 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 very 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,[2] and so we recommend that the second leg be at least a few hops long.

No MIX except the crossover point can distinguish forward messages from replies — even the crossover point cannot be sure whether it is processing a reply or forward message, but it may be able to guess that crossover points are more frequent on forward paths than direct replies or anonymized reply paths.

3.5 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.

[2] For instance, some MIXes may 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.

[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.

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 7.

4 Related design decisions

4.1 Link encryption and what it gets us

Unlike remailer Types I and II that used SMTP [36] (i.e. ordinary Internet e-mail) as their underlying transport mechanism, Mixminion clients and nodes communicate using a forward secure encrypted channel based on TLS [13]. TLS allows the establishment of an encrypted tunnel using ephemeral Diffie-Hellman keys. In order to guarantee that the receiving end is the one intended by the creator of the anonymous message, the receiving node can sign 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 key, and delete the old key material. Key updates don't require any asymmetric encryption techniques, so they are relatively fast.

The purpose of link encryption is to provide forward secrecy: after the 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 decryption notices of past traffic that might be served in some jurisdictions. Even if an attacker manages to get hold of the session key at a particular point they would have to observe all subsequent traffic to be able to update their key appropriately.

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. (See Section 4.2 below.)

4.2 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 4.3), to administer nymservers (see Section 6), 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 public key (for use in header encryption), remixing capability, and batching strategy. MIXes sign these capability blocks and publish them on directory servers (see Section 5). 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 claim that 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.

4.3 Exit policies and abuse

One important entry in a node's capability block is its *exit policy*. 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 let exit from them, such as traffic for local users or other authenticated traffic [41].

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 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.[5] Similarly, if receiving mail is opt-out, an abuser can deny service by forging an opt-out request from a legitimate user. We might instead keep the mail at the exit node and send a note to the recipient telling them how to collect their mail; but this increases server liability by storing messages (see Section 6 below), and also doesn't really solve the problem.

[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 display name and comment parts of an e-mail address could potentially be different for senders who have obtained an address from different sources, leading to smaller anonymity sets.

[5] Compare with the 1965 U.S. Supreme Court case Lamont v. Postmaster General (381 U.S. 301), where the Post Office would detain mail it deemed to be 'communist political propaganda' and instead send a form to the addressee telling him to send back the signed form if he wanted to receive such mail. The government maintained a list of citizens who had filled out these forms.

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, the small number of exit nodes still simplifies traffic confirmation (the adversary observes both a suspected user and the network's exit nodes and looks for timing or packet correlations). The number of available open exit nodes remains a limiting security parameter for the remailer network.

4.4 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 [9], but his solution (which is proposed again in Babel[6]) — 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, legitimate messages may arrive after their expiration date and be dropped. But if we specify expiration dates well after when we expect messages to arrive, messages arriving near their expiration date will be rare: an adversary can delay a message until near its expiration date, then release it and trace it through the network.

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.

A possible compromise solution that still provides forward anonymity is as follows: Messages don't contain any timestamp or expiration information. Each MIX must keep hashes of the headers of all messages it has processed since the last time it rotated its key. MIXes should choose key rotation frequency based on security goals and on how many hashes they want to store.

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. Moreover, if keys overlap, the above delaying attack still

[6] 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.

works. Also note that while key rotation and link encryption (see Section 4.1) 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.

A more complete solution to partitioning attacks may be possible by using the "synchronous batching" approach described in Section 7.2; this is a subject for future research.

5 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 [27]; here we describe Mixminion directory servers and examine the anonymity risks of such information services.

In Mixminion, a group of redundant directory servers serve current node state. It is important that these servers be synchronized and redundant: we lose security if each client has different information about network topology and node reliability. An adversary who controls a directory server can track certain clients by providing different information — perhaps by listing only MIXes it controls or only informing certain clients about a given MIX.

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.[7] Servers can 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.

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 very

[7] New advances in Private Information Retrieval [25] may down the road allow clients to efficiently and privately download a subset of the directory. We recommend against using the MIX-net to anonymously retrieve a random subset: 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.

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. In its basic form this can be very similar to the current ping servers [27], but in the future we can investigate integrating more complex and attack-resistant reputation metrics. Even this reputation information introduces vulnerabilities: for example, an adversary trying to do traffic analysis can get more traffic by gaining a high reputation [14]. We can defend against these attacks by building paths from a suitably large pool of nodes [16] 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 Nym management and single-use reply blocks

Current nymservers, such as `nym.alias.net` [28], maintain a set of (mailbox, reply block) 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. There are 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 reply block for each incoming message. 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 IMAP [11] or POP [33]: 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 nymserver doesn't need to store any reply blocks. The above flooding attack still works, but now it is exactly like flooding a normal IMAP or 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. Hopefully there will be enough volunteers that users can choose the model that makes them most comfortable.

7 Maintaining anonymity sets

7.1 Transmitting large files with Mixminion

We would like to use Mixminion as a transport layer for higher-level applications such as anonymous publication systems [15], but much research remains before we can provide security for users transferring large files over Mixminion.

Alice wants to send a large file to Bob; thus she must send many Mixminion messages. 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 very small fraction of the network can perform this attack, since the Mixminion message size is small.)

Alice seems more likely to maintain her unlinkability by sending all the messages over the same path. On the other hand, a passive adversary can still watch the flood of messages traverse that path. We must hope the honest nodes will hide message streams enough to foil these attacks. The multiple-message tagging attacks described in Section 3.5 make the situation even more dangerous.

A compromise approach is to pick a small number of paths and use them together. Still, if the messages are sent all at once, it seems clear we're going to need some really good cover traffic schemes before we can offer security. The same problem, of maintaining anonymity when sending many messages, comes up when the owner of a pseudonym is downloading his mail from a nymserver.

7.2 Batching Strategy and Network Structure

A MIX-net design groups messages into batches and chooses paths; the approaches it uses affect the degree of anonymity it can provide [3]. We might define ideal anonymity for a MIX-net to be when an attacker can gain no information about the linkage between messages entering and leaving the network, other than that the maximum time between them is equal to the maximum network latency.

This ideal is not achieved by protocols like Mixmaster that use random delays: if the maximum latency of such a network is t, then the anonymity set of a

message leaving the network may be much smaller than all messages that entered over a time t. Also, because Mixmaster is both *asynchronous* (messages can enter and leave the network at any time) and uses free routes, it is subject to the attacks described in [7]. We would like to explore a strategy called *synchronous batching*. This approach seems to prevent these attacks even when free routes are used, and seems to improve the trade-off between latency and anonymity.

The network has a fixed *batch period*, t_{batch}, which is closely related to the maximum desired latency; a typical value could be 3–6 hours. Messages entering the network in each batch period are queued until the beginning of the next period. They are then sent through the MIX-net synchronously, at a rate of one hop per *hop period*. All paths are a fixed length ℓ hops, so that if no messages are dropped, the messages introduced in a given batch will progress through their routes in lock-step, and will all be transmitted to their final destinations ℓ hop periods later. Each subheader of a message specifies the hop period in which it must be received, so that it cannot be delayed by an attacker (which would be fatal for this design).

The latency is between ℓt_{hop} and $t_{\text{batch}} + \ell t_{\text{hop}}$, depending on when the message is submitted. Typically we would have $t_{\text{hop}} < t_{\text{batch}}/\ell$, so the latency is at most $2t_{\text{batch}}$ independent of the path length ℓ.

In practice, several considerations have to be balanced when choosing a batching strategy and network structure. These include maximizing anonymity sets (both batch sizes through each node and the overall anonymity sets of users); bandwidth considerations; reliability; enabling users to choose nodes that they trust; and interactions with the reputation system.

Further analysis is needed before we can choose a final network structure. Note that a planned structure, where each user's software follows the plan consistently when constructing routes, will generally be able to achieve stronger and more easily analyzed security properties than less constrained approaches.

8 Future Directions

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 trickle and flooding attacks [3] as well as intersection attacks on asynchronous free routes [7].
- We need a more thorough investigation of multiple-message tagging attacks, and an analysis of how to safely choose paths when sending many messages.
- We claim that we use conservative techniques, but an all-or-nothing variable-length block cipher is pushing it. Can we keep indistinguishable forward messages and replies using a simpler design?
- We need stronger intuition about how to use dummy messages.

We invite interested developers to join the `mixminion-dev` mailing list [30] and examine the more detailed Mixminion specification [31].

References

1. Masayuki Abe. Universally verifiable MIX with verification work independent of the number of MIX servers. In *Advances in Cryptology - EUROCRYPT 1998, LNCS Vol. 1403*. Springer-Verlag, 1998.
2. Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In *International Workshop on Fast Software Encryption, LNCS*. Springer-Verlag, 1996. <http://citeseer.nj.nec.com/anderson96two.html>.
3. Anonymous. From a trickle to a flood: Active attacks on several mix types. Submitted to Information Hiding Workshop 2002.
4. Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. Proceedings of the Information Hiding Workshop 2001. <http://www.cypherspace.org/adam/pubs/traffic.pdf>.
5. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 115–129. Springer-Verlag, 2000.
6. Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In *Privacy Enhancing Technologies 2002*. Springer-Verlag, 2002.
7. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 30–45. Springer-Verlag, 2000. <http://www.tik.ee.ethz.ch/~weiler/lehre/netsec/Unterlagen/anon/disadvantages_berthold.pdf>.
8. Daniel Bleichenbacher and Anand Desai. A construction of a super-pseudorandom cipher. Manuscript.
9. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982. <http://www.eskimo.com/~weidai/mix-net.txt>.
10. Lance Cottrell. Mixmaster and remailer attacks. <http://www.obscura.com/~loki/remailer/remailer-essay.html>.
11. M. Crispin. Internet Message Access Protocol — Version 4rev1. IETF RFC 2060, December 1996. <http://www.rfc-editor.org/rfc/rfc2060.txt>.
12. Yvo Desmedt and Kaoru Kurosawa. How to break a practical MIX and design a new one. In *Advances in Cryptology - EUROCRYPT 2000, LNCS Vol. 1803*. Springer-Verlag, 2000. <http://citeseer.nj.nec.com/447709.html>.
13. T. Dierks and C. Allen. The TLS Protocol — Version 1.0. IETF RFC 2246, January 1999. <http://www.rfc-editor.org/rfc/rfc2246.txt>.
14. Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. A Reputation System to Increase MIX-net Reliability. Proceedings of the Information Hiding Workshop 2001. <http://www.freehaven.net/papers.html>.
15. Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, July 2000. <http://freehaven.net/papers.html>.
16. Roger Dingledine and Paul Syverson. Reliable MIX Cascade Networks through Reputation. Proceedings of Financial Cryptography 2002. <http://www.freehaven.net/papers.html>.
17. Electronic Frontiers Georgia (EFGA). Anonymous remailer information. <http://anon.efga.org/Remailers/>.

18. D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):39–41, 1999. <http://citeseer.nj.nec.com/goldschlag99onion.html>.
19. 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>.
20. J. Helsingius. anon.penet.fi press release. <http://www.penet.fi/press-english.html>.
21. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999. <http://citeseer.nj.nec.com/jakobsson99flash.html>.
22. Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael 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>.
23. D. Kesdogan, M. Egner, and T. Büschkes. Stop-and-go MIXes providing probabilistic anonymity in an open system. In *Information Hiding Workshop 1998, LNCS Vol. 1525*. Springer Verlag, 1998. <http://www.cl.cam.ac.uk/~fapp2/ihw98/ihw98-sgmix.pdf>.
24. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.
25. Tal Malkin. *Private Information Retrieval*. PhD thesis, MIT, 2000. <http://www.toc.lcs.mit.edu/~tal/>.
26. Tim May. Description of early remailer history. E-mail archived at <http://www.inet-one.com/cypherpunks/dir.1996.08.29-1996.09.04/msg00431.html>.
27. Tim May. Description of Levien's pinging service. <http://www2.pro-ns.net/~crypto/chapter8.html>.
28. David Mazières and M. Frans Kaashoek. The design, implementation and operation of an email pseudonym server. <http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/papers/pnym.txt>.
29. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000. <http://citeseer.nj.nec.com/450148.html>.
30. Mixminion. Mixminion: a type III anonymous remailer. <http://mixminion.net/>.
31. Mixminion. Type III (Mixminion) MIX protocol specifications. <http://mixminion.net/minion-spec.tex>.
32. Ulf Möller and Lance Cottrell. Mixmaster Protocol — Version 2. Unfinished draft, January 2000. <http://www.eskimo.com/~rowdenw/crypt/Mix/draft-moeller-mixmaster2-protocol-00.txt>.
33. 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>.
34. C. Andrew 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>.
35. M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
36. J. Postel. Simple Mail Transfer Protocol. IETF RFC 2821 (also STD0010), August 1982. <http://www.rfc-editor.org/rfc/rfc2821.txt>.

37. Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *ACM Symposium on Theory of Computing*, pages 672–681, 1993. <http://research.microsoft.com/crypto/dansimon/me.htm>.
38. J. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, July 2000. <http://citeseer.nj.nec.com/454354.html>.
39. RProcess. Potato Software. <http://www.skuz.net/potatoware/>.
40. Zero Knowledge Systems. Freedom version 2 white papers. <http://www.freedom.net/info/whitepapers/>.
41. Paul Syverson, Michael Reed, and David 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>.