A Reputation System to Increase MIX-net Reliability

Roger Dingledine[1], Michael J. Freedman[2], David Hopwood[3], David Molnar[4]

[1] Reputation Technologies, Inc. (arma@reputation.com)
[2] Massachusetts Institute of Technology (mfreed@mit.edu)
[3] Independent consultant (david.hopwood@zetnet.co.uk)
[4] Harvard University (dmolnar@hcs.harvard.edu)

Abstract. We describe a design for a reputation system that increases the reliability and thus efficiency of remailer services. Our reputation system uses a MIX-net in which MIXes give receipts for intermediate messages. Together with a set of witnesses, these receipts allow senders to verify the correctness of each MIX and prove misbehavior to the witnesses.

1 Introduction

Anonymous remailers are the most common method of anonymous e-mail communication. Despite wide use of the current global remailer network, this network is generally considered unreliable. Messages are often dropped, and the newsgroup `alt.privacy.anon-server` contains many examples of a message being sent two or three times in the hope that one instance will reach the destination. This unreliability directly affects the number of people using the remailer network and their traffic patterns, which reduces the anonymity these networks provide.

One approach to increasing reliability is to write more reliable software [16]. Another approach is to build MIX protocols that give provable robustness guarantees [3, 7, 13]. Our approach is to build a reputation system to track MIX reliability, and to modify the MIX protocol to support it. Because users choose paths based on the published scores for each MIX, this reputation system improves both reliability (fewer messages get routed through dead MIXes) and efficiency (the system dynamically rebalances the load based on available reliable resources).

Currently deployed remailer reputation systems (better known as *remailer statistics*) collect data independently and are treated as trusted third parties by client software. Reliable statistics servers are good targets for adversaries. Furthermore, these statistics often only measure secondary properties of the MIX-net servers, such as up-time.

We describe related MIX-nets and current statistics systems in Section 2. Section 3 presents a MIX-net design that allows the sender of a message, along with a collection of weakly trusted third party *witnesses*, to prove that a MIX

failed to process a message. Section 4 introduces a new set of agents called *scorers*, who tally failure proofs and serve them to client software. Because each failure can be proven to any number of scorers, loss of a scorer will be less disruptive than loss of a statistics server in current reputation systems.

We stress that this work does not fully solve the problem of MIX reliability. In presenting a simple reputation system, we hope to stimulate further research on reputation systems and modelling the increased reliability they provide.

2 Related Work

2.1 MIX-nets

Chaum introduced the concept of a MIX-net for anonymous communications [2]. A MIX-net consists of a series 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, their order permuted, and forwarded on after stripping the sender's name and identifying information. Chaum also proved security of MIXes against a *passive adversary* who can eavesdrop on all communications between MIXes but is unable to observe the permutation inside each MIX. That is, the *view* of a passive adversary gives negliglible advantage over guessing in linking messages with their senders and receivers.

Current research directions on MIX-nets include "stop-and-go" MIX-nets [8], distributed "flash MIXes" [7] and their weaknesses [3, 13], and hybrid MIXes [14]. Previous work primarily investigates the *robustness* of MIX-nets in the context of a distributed MIX system [7]. A MIX is considered robust if it survives the failure of any k of n participating servers, for some threshold k. This robustness is all or nothing: either k servers are good and the MIX works, or they are not good and the MIX likely will not work.

Robustness has been achieved primarily via zero-knowledge proofs of correct computation. Jakobsson showed how to use precomputation to reduce the overhead of such a MIX network to about 160 modular multiplications per message per server [7], but the protocol was later found to be flawed [13] by Mitsumo and Kurosawa. Desmedt and Kurosawa's alternate approach [3] requires many participating servers. Abe's MIX [1] provides *universal verifiability* in which any observer can determine after the fact whether a MIX cheated, but the protocol is still computationally expensive.

Our notion of reliability differs from robustness in that we do not try to ensure that messages are delivered even when some nodes fail. Instead, we focus on improving a sender's long-term odds of choosing a MIX path that avoids failing nodes. We note that the two approaches can be composed: a distributed MIX with robustness guarantees can be considered as a single node with its own reputation in a larger MIX-net.

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 [6], and by theoretical work on MIXes. Hughes wrote the first cypherpunks anonymous remailer [11]; Finney followed closely with a collection of scripts which used Phil Zimmermann's PGP to encrypt and decrypt remailed messages. Later, Cottrell implemented the Mixmaster system [4], or "Type II" remailers, which added message padding, message pools, and other MIX features lacking in the cypherpunk remailers. At about the same time, Gulcu and Tsudik introduced the Babel system [5], which also created a practical remailer design (although one that never saw widespread use).

2.3 Remailer Statistics

Levien's *statistics pages* [12] track both remailer capabilities (such as what kinds of encryption the remailer supports) and remailer *up-times*, observed 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 [15] allows 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.

2.4 Three Approaches to MIX-Net Reliability

We can build protocols which give specific guarantees of robustness and reliability, and prove theorems about these protocols in suitably specified adversary models. These results may be quite strong, e.g. "this distributed MIX delivers correctly if no more than half of its participating servers are corrupt," yet the resulting protocols may be complicated and inefficient.

Instead of engineering the MIX-net protocol directly to provide reliability, we can make use of reputations to track MIX performance. In this approach, we specify a set of behaviors which characterize a "good" or "bad" MIX. Unlike the reliability via protocol approach, we are not likely to prove strong theorems. The goal of reputation is to make the system "better" without guaranteeing perfection.

A third option is to create economic incentives for MIXes to stay reliable, or ensure that an adversary who wishes to compromise reliability must spend a large amount of resources. Currently, Zero-Knowledge Systems is exploring this approach by paying MIXes for consistent reliability and performance in the Freedom network [17]. Another variant on this approach is to introduce the use of payments for each message sent through a MIX.

Reliability via protocol is the most well-studied approach, while reliability via reputations in the form of Levien statistics is the most widely used. Our work combines the two approaches: we modify the MIX-net protocol to support easy tallying of MIX failures and then specify a suitable reputation system.

3 A MIX-net With Witnessed Failures

Verification of transactions in cryptographic protocols is commonly supported either by performing the transaction publicly, or by using digitally signed receipts. We use the latter notion to develop a design in which MIXes provide a receipt for each message they receive.

The sender Alice can query a MIX to see if it has proof that it attempted to follow the protocol. But the MIX might refuse to provide Alice a receipt for a particular message either because it failed to send the message, or because it was unable to obtain a receipt from the next hop. We solve the problem of pinpointing failures as follows: each message has a *deadline* by which it must be sent to the next MIX. A MIX N_i first tries to send a message directly to the next node, N_{i+1}. If N_i has not received a valid receipt by a specified period before the deadline, it enlists several *witnesses*, who will each independently try to send the message and obtain a receipt. Each witness that obtains a valid receipt sends it back to N_i. Any witness that doesn't will be convinced that N_{i+1} is unreliable, and provides N_i with a signed *statement* to that effect.

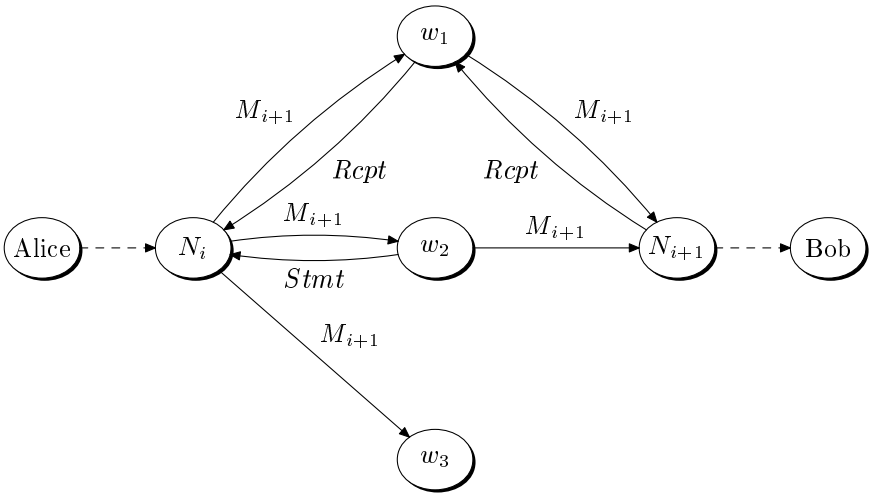

Fig. 1. Message flow example

Thus for each message a MIX sends, it will either have a receipt showing that it processed the message in the time allowed, or statements signed by any witnesses it chooses, asserting that the next MIX did not follow the protocol. We will describe a protocol which uses these receipts and statements to allow the sender of a failed message to demonstrate that a particular node on the path failed.

The above argument does not explicitly consider the possibility that several consecutive MIXes are controlled by an adversary; we cover that in Section 3.5.

While witnesses must be trusted to respond to requests, they need not be trusted to preserve anonymity, since the messages sent to them are no more than the view of a passive adversary with complete network access. Therefore, the proofs of sender and receiver unlinkability that apply to traditional MIX-net protocols still hold.

3.1 Cryptographic Algorithms

We require a public-key encryption scheme, which should be semantically secure under adaptive chosen ciphertext attack, and a public-key signature scheme, which should be existentially unforgeable under adaptive chosen message attack.

The encryption scheme is modelled as a key pair generation algorithm G_E, a randomized encryption algorithm E, and a deterministic decryption algorithm D. The encryption notation explicitly includes the random value: $E_i^r(M)$ means the encryption of message M and random value r under the public key of N_i. We assume that if N_i's key pair is valid (i.e. was generated by G_E), $D_i(E_i^r(M)) = M$ for any plaintext M and random value r.

The signature scheme is modelled as a key pair generation algorithm G_S, a signing algorithm $Sign$, and a verification algorithm Ver. The notation $Sign_i(M)$ means the signature of M under the private key of N_i, and $Ver_i(Sig, M) = 1$ if Sig is a valid signature by N_i on M, or 0 otherwise.

We also assume that authentic, distinct encryption and verification public keys for each MIX are known to all parties. Message recipients are treated as MIXes.

3.2 Overall MIX-net design

Alice wants to send Bob a message anonymously. She chooses a path through the network consisting of $k-1$ MIXes, $N_1 \ldots N_{k-1}$. Alice repeatedly "onion" encrypts her message, and sends the onion to the first MIX in her path. That MIX returns a receipt, processes the onion, and passes the unwrapped-by-one-layer onion to the next MIX, which repeats these steps. If the message does not reach Bob, the transaction has failed. (Section 3.6 shows how to use "end-to-end receipts" to ensure that Alice knows when this has occurred.)

Our system should be able to identify the MIX that caused the failure:

- Goal 1, **Identify failure**: If N_i fails to pass on a well-formed message within the allowed time to the next node N_{i+1}, then Alice can prove to any third party that N_i was the failing MIX (the completeness property).
- Goal 2, **Reject false claims**: No participant (including Alice) can claim that N_i failed to pass on a well-formed message to N_{i+1} in time, unless it really did fail (the soundness property).

3.3 Timing Model

We consider time to be split into periods, each of length one time unit, corresponding to batches of messages sent by MIXes. A message received by N_i in the period ending at time t will have timestamp t on the corresponding receipt signed by N_i. If N_i is not the final recipient, the message must be sent to N_{i+1} in the next period. That is, the *deadline* for sending it to N_{i+1} is $t+1$.

All parties (MIXes, witnesses, senders of messages, and verifiers of failure claims) have clocks that are loosely synchronized (within T_ϵ of the global time). Sending a message over a network is assumed to take T_{comm} in the worst case. Therefore, a party needing to send a message by time t should send it before $t-T_\epsilon-T_{comm}$ according to its local clock, because its local clock may be slow, and because it must allow time for the network communication. A party expecting a message by time t should allow it to be received as late as $t+T_\epsilon$ according to its local clock, since its clock may be fast.

The protocol requires several system-wide time constants:

$T_{response}$ is the time that a MIX is allowed between receiving a message and providing a receipt.

T_{margin} is the length of time before the deadline at which a MIX will attempt to send a message via the witnesses, if it has not been able to obtain a receipt directly from the next node.

T_{retain} is the time (number of periods) for which receipts are retained by MIXes. That is, if a MIX is asked for a receipt in the period ending at t, it need only provide it if the receipt has timestamp $t-T_{retain}$ or later. This value determines the length of time for which failure claims can be directly verified.

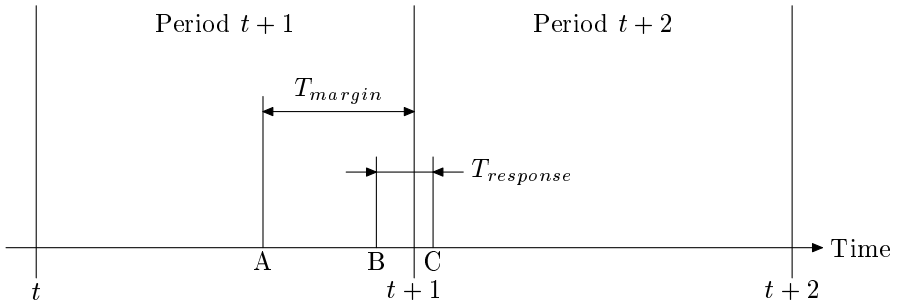

Fig. 2. Timing example

Figure 2 depicts an example time line (showing only global time):

- time A is the point at which N_i stops trying to send to N_{i+1} directly.
- time B is when a witness w tries to contact N_{i+1}.
- time C is the latest time at which N_{i+1} can respond to w with a receipt (this could also be before the deadline t+1).

3.4 Transmitting a Message

This section describes the protocol for transmitting a message from Alice to Bob.

Procedure `Transmit`(*Alice*, *Bob*, *Plaintext*):

1. Alice chooses $k-1$ MIXes $N_1, \ldots N_{k-1}$ to form a "MIX path". Let N_0 be Alice, and let N_k be Bob.
2. Alice picks k random seed values $r_1, r_2, \ldots r_k$.
3. Alice creates an initial packet M_1, defined as

$$M_1 = E_1^{r_1}(N_2, E_2^{r_2}(N_3, \ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots))$$

4. Let *now* be Alice's current local time, and let $Deadline_1 = \lceil now \rceil$.
5. Try to send M_1 and $Deadline_1$ directly to N_1, waiting for a receipt.
6. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } M_1, Deadline_1\text{"}) = 1$; if so, stop.
7. If no receipt is returned, set $Deadline_1 := Deadline_1 + 1$, and use the procedure `Hop-send`$(N_1, M_1, Deadline_1)$ below to resend M_1 to N_1.

If all parties follow the protocol, the message will then be processed by $N_1, \ldots N_{k-1}$ as follows:

- N_1 reads M_1 from Alice, and processes it according to the procedure `Hop-receive`$(Alice, N_1, M_1, Deadline_1)$.
- N_1 decrypts M_1 to give (N_2, M_2), where

$$M_2 = E_2^{r_2}(N_3, \ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots)$$

- Let $Deadline_2 = Deadline_1 + 1$.
- N_1 uses the procedure `Hop-send`$(N_2, M_2, Deadline_2)$ to send M_2 to N_2.
- This process is repeated by N_2, which sends M_3 to N_3, and so on for $N_3, N_4, \ldots N_{k-1}$.
- Eventually, $E_k^{r_k}(Plaintext)$ is sent to Bob. Bob can decrypt this message, so the plaintext has successfully been transmitted.

Procedure `Hop-send`(N_{dest}, *Message*, *Deadline*):

1. Try to send $Message$ and $Deadline$ to N_{dest} directly, waiting for a receipt.
2. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$.
3. If a valid receipt is not received before $Deadline - T_{margin} - T_\epsilon$ (by the sending node's local clock),
   (a) Let W be a set of witnesses.
   (b) Send "Witness: $N_{dest}, Message, Deadline$" to each $w \in W$ (causing `Witness` to be called on each w). Wait for any w to send back a receipt.

(c) If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$.
(d) If no valid receipt is received, store any statements returned by the witnesses.

Note that an imposter witness or MIX may send fake receipts to the sender in order to try to confuse it. The sender should ignore any receipts that are not valid. If the sender receives more than one valid receipt, it need only store one (chosen arbitrarily).

Procedure `Hop-receive`$(N_{src}, N_{dest}, Message, Deadline)$**:**

1. Let *now* be the current time.
2. If $now > Deadline + T_\epsilon$ or $now < Deadline - 1 - T_\epsilon$, drop the message and respond with an error.
3. Otherwise, decrypt $Message$, and queue for transmission in the next period by `Hop-send`.
4. Send back the receipt $Sign_{dest}(\text{"Receipt: } Message, Deadline\text{"})$ to N_{src}.

Procedure `Witness`$(N_{src}, N_{dest}, Message, Deadline)$**:**

1. Let *now* be the current time.
2. (The witness must be sure that N_{dest} has time to respond:) If $now > Deadline - T_{comm} - T_\epsilon$ or $now < Deadline - 1 + T_\epsilon$, drop the message and respond with an error.
3. Try to send $Message$ to N_{dest} directly, waiting for a receipt.
4. If a receipt *Rcpt* is received, check that $Ver_{dest}(Rcpt, \text{"Receipt: } Message, Deadline\text{"}) = 1$. If so, send it back to N_{src}.
5. If a valid receipt is not received before $Deadline + T_{response} + T_\epsilon$, N_{dest} has failed; send back a statement "Failed: $N_{dest}, Message, Deadline$", signed by the witness, to N_{src}.

3.5 Identifying and Proving Failed Transactions

Alice can prove to any chosen verifier (call him Victor) a claim that a MIX failed to deliver a message. Section 4 describes Victor's duty in our reputation system.

Suppose that N_i $(1 \leq i < k)$ is the first node on the path that does not follow the protocol, i.e., it fails to handle a message M_i that N_{i-1} sends to it, within the allowed time. Alice wants to prove to Victor that N_i failed to process this message (which she might discover by a binary search over her MIX path).

Because N_{i-1} behaved according to protocol, it will have a receipt $Rcpt_i$ for the message M_i with deadline $Deadline_i$. As Alice knows all the random padding values, she can compute the message that N_i is supposed to send to N_{i+1}:

$$(N_{i+1}, M_{i+1}) = D_i(M_i) = (N_{i+1}, E_{i+1}^{r_{i+1}}(\ldots E_{k-1}^{r_{k-1}}(Bob, E_k^{r_k}(Plaintext))\ldots))$$

She suspects that M_{i+1} was not sent to N_{i+1}, so she prepares a claim by obtaining the $Rcpt_i$ from N_{i-1}, and calculating M_{i+1} as above. Then Alice sends "I blame N_i, claim: $r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i$", which Victor verifies:

Procedure `Verify-claim`$(N_i, r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i)$**:**

1. Check that N_i and N_{i+1} refer to valid MIXes, and M_{i+1} is the correct length for an intermediate message.
2. Calculate $M_i = E_i^{r_i}(N_{i+1}, M_{i+1})$, and check that $Ver_i(Rcpt_i, \text{"Receipt: } M_i, Deadline_i\text{"}) = 1$.
3. Let *now* be the current time. If $Deadline_i + 1 < now - T_{retain} + T_\epsilon$, then it is too late for the claim to be verified, since N_i may legitimately have discarded its receipt; the claim is therefore rejected.
4. Send "Receipt request: N_{i+1}, M_{i+1}" to N_i.
5. If "Receipt response: $Rcpt_{i+1}, Timestamp$" is received from N_i within time $T_{response}$, such that $Ver_{i+1}(Rcpt_{i+1}, \text{"Receipt: } M_{i+1}, Timestamp\text{"}) = 1$ and $Timestamp \leq Deadline_i + 1$, then reject the claim.
6. If statements "Failed: $N_{i+1}, M_{i+1}, Timestamp$" signed by a sufficient set of witnesses are received, for some $Timestamp \leq Deadline_i + 1$, conclude that N_i made a reasonable attempt to send the message, and reject the claim. (See Section 3.7 for discussion of trust requirements on witnesses.)
7. Otherwise, conclude that N_i failed – either because it did not process the original message, or because it did not respond to the receipt request.

Proving that a delivery failure results in a proper claim: N_{i+1} will only give out a receipt for which $Ver_{i+1}(Rcpt_{i+1}, \text{"Receipt: } M_{i+1}, Deadline_i + 1\text{"}) = 1$ if it received the message M_{i+1} by time $Deadline_i + 1$. If it did not receive M_{i+1} by then, assuming N_{i+1}'s signatures cannot be forged, N_i will not be able to provide such a signature. Thus, Victor will conclude that it failed.

Node N_{i+1} might be controlled by the adversary; in this case, it might provide a receipt on M_{i+1} in order to exonerate N_i, even though M_i was not actually processed. However, Alice can then use this receipt in the same protocol to attempt to prove that N_{i+1} failed. Therefore, the adversary will be able to choose which of the contiguous MIXes it controls can be proven unreliable. However, since there are only $k - 2$ other nodes on the path, Alice will be able to prove that some node failed, after at most $k - 1$ iterations of the protocol.

Because Alice can always prove that some MIX failed if it did in fact fail, we satisfy Goal 1, being able to identify failures in the MIX-net.

We note that knowledge of the fact that N_i and N_{i+1} are part of Alice's chosen path is *not* part of the view of a passive adversary in a normal MIX-net protocol. The closer N_{i+1} is to the end of the path, the more likely it is that this additional information will allow an adversary to link Alice with Bob. To avoid giving away this information, Alice may send all the messages needed for the failure proof over the MIX-net. When requesting receipts she can include a reply block, so that the node will be able to send back the receipt while Alice remains anonymous.

It may seem circular to rely on the MIX-net to send messages needed for a failure proof, when the goal of the proof protocol is to improve MIX-net reliability. However, since these messages are only used to prove failure and do not convey any other information, it is not critical that all of them are successfully

transmitted. Alice can repeat any attempts to obtain receipts and to send the claim message to Victor as often as necessary, using random independent paths for each attempt. This will succeed quickly provided the probability of a message making it through the MIX-net (from Alice) is not too small.

Proving that false claims are rejected: We wish to show that no participant can claim that N_i failed to pass on a well-formed message sent by Alice to N_{i+1}, unless it really did fail to send such a message. Without loss of generality, we will consider the adversary to be Alice.

Recall that Alice's claim to Victor is of the form "I blame N_i, claim: $r_i, N_{i+1}, M_{i+1}, Deadline_i, Rcpt_i$". Victor then calculates $M_i = E_i^{r_i}(N_{i+1}, M_{i+1})$. Decrypting both sides, we obtain $D_i(M_i) = (N_{i+1}, M_{i+1})$, assuming N_i's key pair is valid.[1]

- Suppose Alice caused the message M_i to be sent to N_i by time $Deadline_i$, and then tried to claim that N_i failed. A well-behaving N_i will decrypt M_i to give the next hop and intermediate message (N_{i+1}, M_{i+1}), and try to send M_{i+1} to N_{i+1} using `Hop-send`. Either N_i will obtain a receipt for this message (signed by N_{i+1} and having the timestamp $Deadline_i + 1$), that refutes Alice's claim, or it will have signed statements from a sufficient set of witnesses (see Section 3.7) saying that N_{i+1} refused to provide a receipt when it was obliged to do so. In either case, N_i will be exonerated.
- Suppose Alice did not cause M_i to be sent to N_i by $Deadline_i$.
  In order to make a credible claim, Alice needs a receipt $Rcpt_i$ such that $Ver_i(Rcpt_i, \text{"Receipt: } M_i, Deadline_i\text{"}) = 1$ (since Victor will check this). However, if N_i did not receive M_i, it will not have given out any such receipt, and so assuming that N_i's signatures cannot be forged,[2] it will be exonerated.

A node could also be falsely accused of failure if there are "sufficient" witness statements against it as defined in Section 3.7. We assume that this does not occur, because the adversary is not able to subvert witnesses in the core group.

Therefore, we satisfy Goal 2. Note that the above argument covers the case in which Alice attempts to send messages to N_i that either do not decrypt, or decrypt to ill-formed plaintexts, since we have proven that for Alice's claim to be accepted, N_i must have received a message with a well-formed (N_{i+1}, M_{i+1}) pair as the plaintext.

Many MIX-net protocols require MIXes to refuse to pass on a message if it is a *replay* — a message the MIX had already received within a given time period (see [5] for a rationale). We prevent MIXes from losing reputation because of this behavior as follows. When a MIX receives a replayed message (using `Hop-receive` as normal), it will already have a receipt from the previous time it

[1] We can assume that N_i's public key is valid because it is chosen by N_i, who could only hurt its own reputation via an invalid key.

[2] We take the position that if N_i's private key has been compromised, it should be considered to have failed.

sent that message (or else witness statements showing that it tried to send it). Steps 5 and 6 of `Verify-claim` show that it can provide the earlier receipt or statements when challenged with a failure claim. We define the length of time for which replays are remembered to be the same as T_{retain}. A MIX that receives a replayed message should delay the expiration of the earlier receipt or statements for a further T_{retain} periods.

3.6 End-to-End Receipts

If Alice is posting to a public forum like Usenet, she (and any verifier) can see whether the message was posted correctly. Otherwise, if the MIX-net supports reply blocks or another method for two-way communication, Bob's software can immediately send an *end-to-end receipt* back through the MIX-net to Alice. Since a failure can also occur on the return path, Alice should be able to prove one of three cases: a MIX on the forward path failed; a MIX on the reply path failed; or Bob failed to send a receipt.

With careful design of message formats, it is possible to make receipt messages indistinguishable from forward messages to nodes on the reply path. In that case, the same protocol used to claim failures in sending forward messages will be applicable to reply messages.

3.7 Trust Requirements for Witnesses

Dishonest witnesses do not compromise anonymity, but they can compromise reliability. Witnesses could either refuse to give a statement or receipt for a MIX that has failed, or (especially if users trust an arbitrary set of witnesses chosen by a sending node) make false statements in order to frame a MIX that has not in fact failed. We suggest defining a core group of witnesses who are relatively widely trusted. If some threshold number of this core group provide statements implying that a node N_{i+1} has failed, that would be considered *sufficient* for the purposes of `Verify-claim`. Specifying a fixed (or even slowly changing) group of witnesses is not ideal, but if the messages sent to this group are also published, other parties can duplicate their actions and gain confidence in their behavior.

4 Reputation Systems

Reputations have been suggested as a means of improving MIX-net reliability [10, 12]. In layering on a reputation system, we add two more agents to the system. *Raters* make observations about the performance or honesty of MIXes. In our case, the raters are both the sender Alice and any MIXes that make use of the witnesses. *Scorers* tally observations from raters and make these tallies (or *scores*) available. For simplicity, we choose to give the scorer the duties of verifier and witness as well.

As in [15], sender software must be configurable to automatically use scores. Any user of the MIX-net must be able to contribute *ratings*. The scoring system

must be verifiable: scorers can determine the credibility of ratings, and other users can verify that scorers are tallying ratings correctly. Clients must be able to draw conclusions from scores that lead to "good" predictions. The overall scoring algorithm must be dynamic, recognizing and reflecting recent trends in MIX performance. While achieving all of these goals, the system must also maintain the level of anonymity provided by the MIX-net.

4.1 Reputation System Overview

We introduce a set of scorers, each named Sally. Each Sally keeps her own database of performance scores for MIXes. She receives, verifies, and tallies failure claims. Sally also sends test messages to distinguish reliable MIXes (few delivery failures due to good performance) from new MIXes (few delivery failures because nobody has tried them yet).

If we simply count the number of messages that each MIX drops, an effective attack would be to constantly add new unreliable MIXes. Therefore scores include both a count of negative ratings, and also a "minimum number of positive ratings" requirement, which is a threshold configurable on the client side. Client software downloads Sally's reputation database, and allows the user to specify parameters when selecting acceptable MIX paths, such as "expected success of transmission".

If our assumptions in Section 3.5 hold, there is no way to spoof negative ratings. Note that an adversary may be able to force negative ratings on a MIX, while goal 2 still holds: if he floods that MIX's incoming bandwidth (either directly or via witnesses), the MIX will no longer be able to sustain the load. However, this is exactly the point where the MIX demonstrates that it is unreliable. Causing MIXes to lose reputation in the face of *successful* flooding is consistent with our scoring system goals: the scoring system measures reliability and capabilities, not intent.

4.2 Increasing Confidence in Positive Ratings

An adversary can easily fake a positive rating by building a message which uses a MIX path entirely owned by him, and then generating a transcript which "proves" successful transmission. We need to make positive ratings actually reflect a MIX's ability to successfully deliver Alice's message in the future.

One approach to making positive ratings more reliable (and thus more meaningful) is to build a graph based on rater credibility such as that employed by Advogato [9]. Similar to the PGP web of trust, this metric aims to reduce the damage that an adversary can cause by *pseudospoofing* – creating a multitude of identities each controlled by that adversary. Another approach is to treat reputation as a probability: an estimate of the expected outcome of a transaction with that MIX. Scores might simply be the sum of ratings, normalized and weighted by the credibility of raters. Other designs employ neural networks or data clustering techniques to apply non-linear fitting and optimization systems to the field of reputation.

Our solution emphasizes simplicity and ease of implementation and evaluation. We solve the positive rating credibility problem by having each Sally produce positive ratings herself — after all, if Sally sends the test messages herself, she knows they are unbiased ratings. MIXes that process her message will earn positive ratings that Sally knows to be accurate.

It may be possible for an adversary to guess whether a message has been received directly from a sender (i.e. the adversary is the first hop on the path, N_1), or if it is being sent to the final recipient (i.e. the adversary is N_{k-1}). Unfortunately, it is difficult to produce simulated messages that are completely indistinguishable from real messages in these cases. We do not have a fully satisfactory solution to this for positive ratings; instead, we rely on negative ratings to expose an adversary that behaves unreliably only when it is the first or last hop.

Sally should expire her memories of transactions after a certain amount of time so that old failures do not haunt a MIX forever. Similarly, MIXes need to have a threshold of recent successes in order to stay "in the running". Alice configures her client software to choose only MIXes that have some minimum number of positive ratings. Out of this pool, she weights the MIXes she uses for her path based on the number of verified delivery failures observed for this MIX.

This system reacts quickly to a decrease in reliability of a MIX. A MIX with high reputation will have many users routing messages through it. if it suddenly stops delivering messages, these users will quickly deliver a series of negative ratings. This negative feedback process serves to stabilize the system so scores reflect reality.

For redundancy and to allow verifiability of scorers, Alice can remember which mails had a corresponding end-to-end receipt, tally her transactions, and build her own score table in which she is confident of both the positive ratings and the negative ratings. Periodically comparing her score tables with those of the available Sally's allows Alice to "test the waters" herself and weakens the trust requirements on scorers. If claims for dropped messages are published, anybody can verify them. Thus Alice might keep track of negative ratings for a few weeks, then compare with Sally to determine if Sally's scores are actually reflecting all of the negative ratings.

4.3 Implications for Traffic Analysis

The addition of witnesses to the protocol may introduce new attacks. Direct communications between nodes can use link encryption, but encrypting the messages to witnesses would have little benefit (and would be incompatible with publishing these messages, as suggested in Section 3.7). So if an adversary can force the witnesses to be used instead of direct communication, this may weaken the security of the network.

The reputation system also introduces new attacks. Eve could gain a high reputation and thus get more traffic routed through her MIX, in order to make traffic analysis easier. In a system without reputations, the way to purchase more traffic to analyze is not so clear; now it is simply a matter of maintaining a reliable

MIX. In addition, the adversary now has incentive to degrade or sabotage the performance of other nodes to make his relative reputation higher. This kind of attack was described by RProcess as "selective denial of service": the bad guys want traffic to go through their nodes, so they ensure that all other nodes are less reliable [16]. As recent distributed denial of service attacks demonstrate, crippling an Internet host can be easy. Scorers must expire ratings promptly enough to prevent an adversary from easily tarnishing the reputations of all other MIXes; this system tuning will be extremely complex and difficult.

On the other hand, we may be making the system *more* anonymous by making it more reliable. Currently, users may have to send a message many different times, through many different paths, before it gets through. These re-sends of the same message offer more information to traffic analyzers. A better theoretical framework for traffic analysis needs to be established before we can make any quantifiable statements about the implications of the proposed protocol.

5 Conclusion and Future Directions

We have described a reputation system for remailer networks, based on a MIX-net design that employs receipts for intermediate messages. There are a number of directions for future research:

- Create a "reliability metric" and an accompanying model which will allow us to quantify the behavior of our reputation system. For instance, we might consider as a metric the expected probability of a message making it from sender to receiver. Then we would calculate reliability with and without the reputation system in place and see whether reliability improves. A parallel model characterizing efficiency of a MIX network might be very enlightening, especially from a network flow optimization viewpoint.
- Can we achieve some further measure of reliability (or resource management) through the use of electronic cash or similar accountability measures?
- Can we defend against the selective DoS attack described in Section 4.3? How can we protect against adversaries who want their MIXes listed as most reliable?
- Can we make a reputation system that is both efficient and universally verifiable? Currently, only Alice can prove that a message did not reach its destination. Can we apply zero-knowledge proofs so that Alice does not leak any information about the next hop, while remaining practical? Can we extend this so that anyone can detect a failed MIX?

This paper provides a foundation for further analysis of MIX-net reliability and reputations. Our reputation system is designed to be simple and easily extensible. Much work remains in a wide variety of directions before a reliable, secure, and ubiquitous remailer network can be put in place.

Acknowledgements

We thank Nick Mathewson and Blake Meike for help with the reputation system; Nick Feamster, Kevin Fu, Chris Laas, Anna Lysyanskaya, and Marc Waldman for discussions; and our anonymous reviewers for many useful comments.

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. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982.
3. 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.
4. Electronic Frontiers Georgia (EFGA). Anonymous remailer information. <http://anon.efga.org/Remailers/>.
5. C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium - NDSS '96*. IEEE, 1996.
6. J. Helsingius. `anon.penet.fi` press release. <http://www.penet.fi/press-english.html>.
7. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999.
8. 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.
9. Raph Levien. Advogato's trust metric. <http://www.advogato.org/trust-metric.html>.
10. Tim May. Cyphernomicon. <http://www2.pro-ns.net/~crypto/cyphernomicon.html>.
11. 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>.
12. Tim May. Description of Levien's pinging service. <http://www2.pro-ns.net/~crypto/chapter8.html>.
13. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
14. M. Ohkubo and M. Abe. A Length-Invariant Hybrid MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000.
15. RProcess. Potato Software. <http://www.skuz.net/potatoware/>.
16. RProcess. Selective denial of service attacks. <http://www.eff.org/pub/Privacy/Anonymity/1999_09_DoS_remail_vuln.html>.
17. Zero Knowledge Systems. Freedom version 2 white papers. <http://www.freedom.net/info/whitepapers/>.