No Right to Remain Silent: Isolating Malicious Mixes

Hemi Leibowitz, ***Bar-Ilan University, IL;*** **Ania M. Piotrowska and George Danezis,**
University College London, UK; **Amir Herzberg,** ***University of Connecticut, US***

https://www.usenix.org/conference/usenixsecurity19/presentation/leibowitz

This paper is included in the Proceedings of the
28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the
28th USENIX Security Symposium
is sponsored by USENIX.

No Right to Remain Silent: Isolating Malicious Mixes

Hemi Leibowitz
Bar-Ilan University, Israel

Ania M. Piotrowska
University College London, UK

George Danezis
University College London, UK

Amir Herzberg
University of Connecticut, US

Abstract

Mix networks are a key technology to achieve network anonymity and private messaging, voting and database lookups. However, simple mix network designs are vulnerable to malicious mixes, which may drop or delay packets to facilitate traffic analysis attacks. Mix networks with provable robustness address this drawback through complex and expensive proofs of correct shuffling but come at a great cost and make limiting or unrealistic systems assumptions. We present *Miranda*, an efficient mix-net design, which mitigates active attacks by malicious mixes. *Miranda* uses both the detection of corrupt mixes, as well as detection of faults related to a pair of mixes, without detection of the faulty one among the two. Each active attack – including dropping packets – leads to reduced connectivity for corrupt mixes and reduces their ability to attack, and, eventually, to detection of corrupt mixes. We show, through experiments, the effectiveness of *Miranda*, by demonstrating how malicious mixes are detected and that attacks are neutralized early.

1 Introduction

The increasing number of bombshell stories [27, 19, 10] regrading mass electronic surveillance and illicit harvesting of personal data against both ordinary citizens and high-ranking officials, resulted in a surge of anonymous and private communication tools. The increasing awareness of the fact that our daily online activities lack privacy, persuades many Internet users to turn to encryption and anonymity systems which protect the confidentiality and privacy of their communication. For example, services like WhatsApp and Signal, which offer protection of messages through end-to-end encryption, gained popularity over the past years. However, such encryption hides only the content but not the meta-data of the message, which carries a great deal of privacy-sensitive information. Such information can be exploited to infer who is communicating with whom, how often and at what times. In contrast, the circuit-based onion routing Tor network is an example of anonymity systems that protect the meta-data. Tor is currently the most popular system offering anonymity, attracting almost 2 million users daily. However, as research has shown [47, 40, 42, 41], Tor offers limited security guarantees against traffic analysis.

The need for strong anonymity systems resulted in renewed interest in *onion mixnets* [12]. In an onion mixnet, sender encrypts a message multiple times, using the public keys of the destination and of multiple mixes. Onion mixnets are an established method for providing provable protection against meta-data leakage in the presence of a powerful eavesdropper, with low computational overhead. Early mixnets suffered from poor scalability, prohibitive latency and/or low reliability, making them unsuitable for many practical applications. However, recently, researchers have made significant progress in designing mixnets for high and low latency communication with improved scalability and performance overhead [44, 48, 13]. This progress is also visible in the industrial sector, with the founding of companies whose goal is to commercialise such systems [1, 2].

Onion mixnets offer strong anonymity against passive adversaries: a single honest mix in a cascade is enough to ensure anonymity. However, known mixnet designs are not robust against *active* long-term traffic analysis attacks, involving *dropping* or *delaying* packets by malicious mixes. Such attacks have severe repercussions for privacy and efficiency of mix networks. For example, a disclosure attack in which a rogue mix strategically drops packets from a specific sender allows the attacker to infer with whom the sender is communicating, by observing which recipient received fewer packets than expected [4]. Similarly, Denial-of-Service (DoS) attacks can be used to enhance de-anonymization [9], and $(n-1)$ attacks allow to track packets over honest mixes [45].

It is challenging to identify and penalize malicious mixes while retaining strong anonymity and high efficiency. Trivial strategies for detecting malicious mixes are fragile and may become vectors for attacks. Rogue mixes can either hide their involvement or worse, make it seem like honest mixes are unreliable, which leads to their exclusion from the net-

work. Several approaches to the problem of active attacks and reliability were studied, however, they have significant shortcomings, which we discuss in Section 8.

In this work, we revisit the problem of making decryption mix networks robust to malicious mixes performing active attacks. We propose *Miranda*[1], an efficient reputation-based design, that detects and isolates active malicious mixes. We present security arguments that demonstrate the effectiveness of Miranda against active attacks. The architectural building blocks behind Miranda have been studied by previous research, but we combine them with a novel approach which takes advantage of detecting failure of inter-mix links, used to isolate and disconnect corrupt mixes, in addition to direct detection of corrupt mixes. This allows Miranda to mitigate corrupt mixes, without requiring expensive computations.

Miranda disconnects corrupt mixes by carefully gathering evidence of their misbehavior, resulting in the removal of links which are misused by the adversary. The design includes a set of secure and decentralized *mix directory authorities* that select and distribute mix cascades once every *epoch*, based on the gathered evidence of the faulty links between mixes. Repeated misbehaviors result in the complete exclusion of the misbehaving mixes from the system (see Figure 1).

We believe that Miranda is an important step toward a deployable, practical strong-anonymity system. However, Miranda design makes several significant simplifying assumptions. These include (1) a fixed set of mixes (no churn), (2) a majority of benign mixes (no Sybil), (3) reliable communication and efficient processing (even during DoS), and (4) synchronized clocks. Future work should investigate, and hopefully overcome, these challenges; see Section 9.

Contributions. Our paper makes the following contributions:

- We present Miranda, an efficient, low-cost and scalable novel design that detects and mitigates active attacks. To protect against such attacks, we leverage reputation and local reports of faults. The Miranda design can be integrated with other mix networks and anonymous communication designs.
- We propose an encoding for secure *loop messages*, that may be used to securely test the network for dropping attacks – extending traditional mix packet formats for verifiability.
- We show how Miranda can take advantage of techniques like community detection in a novel way, which further improves its effectiveness.
- We analyze the security properties of Miranda against a wide range of attacks.

[1] "Miranda warning" is the warning used by the US police, in order to notify people about their rights before questioning them. Since Miranda prevents adversaries from silently (but actively) attacking the mix network, we refer to it as *no right to remain silent*.

Overview. The rest of this paper is organized as follows. In Section 2, we discuss the motivation behind our work, and define the threat model and security goals. In Section 3, we present important concepts of Miranda. In Sections 4 and 5, we detail the core protocols of Miranda, which detect and penalize active attacks. In Section 6, we present improved, *community-based* detection of malicious mixes. In Section 7, we evaluate the security properties of Miranda against active attacks. In Section 8, we contrast our design to related work. Finally, we discuss future work in Section 9 and conclude in Section 10.

2 The Big Picture

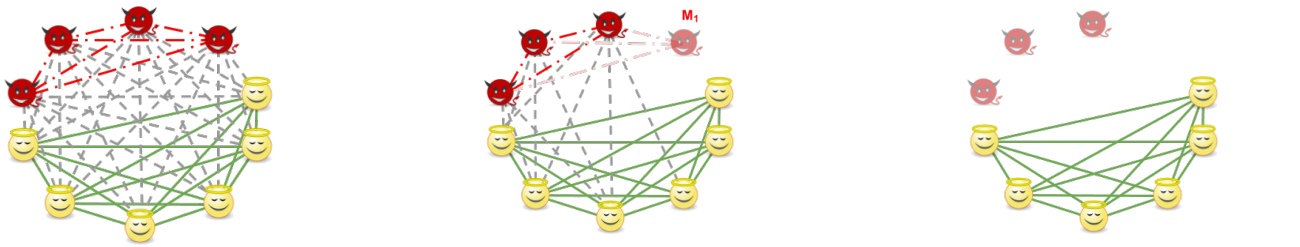
In this section, we outline the general model of the Miranda design, define the threat model, and motivate our work by quantifying how active attacks threaten anonymity in mix networks. Then, we summarize the security goals of Miranda.

2.1 General System Model

We consider an anonymous communication system consisting of a set of users communicating over the *decryption mix network* [12] operating in synchronous batches, denoted as *rounds*. Depending on the path constraints, the topology may be arranged in separate cascades or a Stratified network [21]. We denote by $\mathcal{M}$ the set of all mixes building the anonymous network. For simplicity, in this work we assume that the set of mixes $\mathcal{M}$ is fixed (no churn). See discussion in Section 9 of this and other practical challenges.

Messages are end-to-end *layer encrypted* into a cryptographic packet format by the sender, and the recipient performs the last stage of decryption. Mixes receive packets within a particular round, denoted by r. Each mix decodes a successive layer of encoding and shuffles all packets randomly. At the end of the round, each mix forwards all packets to their next hops. Changing the binary pattern of packets by removing a single layer of encryption prevents bit-wise correlation between incoming and outgoing packets. Moreover, mixing protects against an external observer, by obfuscating the link between incoming and outgoing packets.

Message packet format. In this paper, we use the Sphinx cryptographic packet format [15]. However, other packet formats can be used, as long as they fulfill certain properties. The messages encoded should be of *constant length* and *indistinguishable* from each other at any stage in the network. Moreover, the encryption should guarantee *duplicates detection*, and eliminate tampered messages (*tagging attacks*). The packet format should also allow senders to encode arbitrary routing information for mixes or recipients. We denote the result of encoding a message as $\mathsf{Pack}(\mathsf{path}, \mathsf{routingInfo}, \mathsf{rnd}, \mathsf{recipient}, \mathsf{message})$, where rnd



(a) Connectivity graph in the beginning. All mixes are willing to communicate with each other.

(b) Miranda detects active attacks and removes the links between the honest and dishonest nodes (Section 4.3).

(c) Miranda applies community detection (Section 6) to further detect dishonest nodes and disconnect them from the honest nodes.

Figure 1: High-level overview of the process of isolating malicious mixes in Miranda.

denotes a random string of bits used by the packet format.

2.2 Threat Model

We consider an adversary whose goal is to de-anonymize packets traveling through the mix network. Our adversary acts as a *global observer*, who can eavesdrop on all traffic exchanged among the entities in the network, and also, knows the rate of messages that Alice sends/receives [2]. Moreover, all malicious entities in the system collude with the adversary, giving access to their internal states and keys. The adversary may control many participating entities, but we assume a majority of honest mixes and *directory servers* (used for management, see Section 3). We allow arbitrary number of malicious clients but assume that there are (also) many honest clients - enough to ensure that any first-mix in a cascade, will receive a 'sufficient' number of messages in most rounds - say, 2ω, where ω is sufficient to ensure reasonable anonymity, for one or few rounds.

In addition, Miranda assumes reliable communication between any pair of honest participants, and ignores the time required for computations - hence, also any potential for Miranda-related DoS. In particular, we assume that the adversary cannot arbitrarily drop packets between honest parties nor delay them for longer than a maximal period. This restricted network adversary is weaker than the standard Dolev-Yao model, and in line with more contemporary works such as XFT [35] that assumes honest nodes can eventually communicate synchronously. It allows more efficient Byzantine fault tolerance schemes, such as the one we present. In practice, communication failures *will* occur; see discussion in Section 9 of this and other practical challenges.

We denote by n the total number of mixes in the network ($|\mathcal{M}| = n$), n_m of which are malicious and n_h are honest ($n = n_m + n_h$). We refer to cascades where all mixes are malicious as *fully malicious*. Similarly, as *fully honest* we refer to cascades where all nodes are honest, and *semi-honest* to the ones where only some of the mixes are honest. A link between an honest mix and a malicious mix is referred to as a *semi-honest* link.

[2] We emphasize that this is a non-trivial adversarial advantage. In reality, the adversary might not know Alice's rate, and therefore might be more limited regarding de-anonymization attacks.

2.3 What is the Impact of Active Attacks on Anonymity?

Active attacks, like dropping messages, can result in a catastrophic advantage gained by the adversary in linking the communicating parties. To quantify the advantage, we defined a security game, followed by a qualitative and composable measure of security against dropping attacks. To our knowledge, this is the first analysis of such attacks and we provide full details in [34]. Our results support the findings of previous works on statistical disclosure attacks [4] and DoS-based attacks [9], arguing that the traffic analysis advantage gained from dropping messages is significant. We found that the information leakage for realistic volumes of traffic (10–100 messages per round) is quite significant: *the adversary can improve de-anonymization by about 20%*. For larger traffic rates (more than 1000 messages per round) the leakage drops but expecting each client to receive over 1000 messages per round on average seems unrealistic, unless large volumes of synthetic cover traffic is used. The lesson drawn from our analysis and previous studies is clear: it is crucial to design a mechanism to detect malicious nodes and remove them from the system after no more than a few active attacks. The Miranda design achieves this goal.

2.4 Security Goals of Miranda

The main goal of a mix network is to hide the correspondence between senders and recipients of the messages in the network. More precisely, although the communication is over cascades that might contain malicious mixes, the Miranda design aims to provide protection which is *indistinguishable from the protection provided by an 'ideal mix'*, i.e., a single mix node which is known to be honest.

The key goals of Miranda relate to alleviating and discouraging active attacks on mix networks, as they have a significant impact on the anonymity through traffic analysis. This is achieved through the detection and exclusion of misbehaving mixes. The Miranda design offers the following protections against active attacks:

Detection of malicious nodes. Every active attack by a corrupt mix is detected with non-negligible probability, by at least one entity.

Separation of malicious nodes. Every active attack by a rogue mix results, with a non-negligible probability, in the removal of at least one link connected to the rogue mix - or even removal of the rogue mix itself.

Reducing attacks impact over multiple epochs. Repeated application of Miranda lowers the overall prevalence and impact of active attacks by corrupt mixes across epochs, limiting the ability of the adversary to drop or delay packets.

3 Rounds, Epochs and Directories

In Miranda, as in other synchronous mixnet designs, time is broken into *rounds*, and in each round, a mix 'handles' all messages received in the previous round. However, a more unique element of Miranda is that rounds are collected into *epochs*. Epochs are used to manage Miranda; the beginning of each epoch includes announcement of the set of cascades to be used in this epoch, after a selection process that involves avoidance of mixes detected as corrupt - and of links between two mixes, where one or both of the mixes reported a problem.

The process of selecting the set of cascades for each epoch, is called the *inter-epoch process*, and is performed by a set of d servers refered to as *directory authorities*, following [14], which maintain a list of available mixes and links between them. We assume that a number d_m of authorities can be malicious and collude with the adversary or deviate from the protocol, in order to break the security properties. By d_h we denote the number of honest authorities $(d = d_m + d_h)$, which follow the protocol truthfully.

During each epoch, there are multiple rounds where users communicate over the mix network. Both users and mixes report any misbehavior they encounter to the directory authorities. The *directory authorities* process these reports, and, before the beginning of a new epoch, they select a set of cascades available in that epoch. The newly generated cascades will reflect all reported misbehaviors. Namely, cascades exclude links that were reported, or mixes involved in too many reports, or detected via Miranda's *community-based attacker detection mechanisms*, described in Section 6.

We denote the number of reports which marks a mix as dishonest and causes its exclusion from the network as *thresh* and emphasize that *thresh* is cumulative over rounds and even epochs. In this paper, we simply use $thresh = n_m + 1$, which suffices to ensure that malicious mixes cannot cause Miranda to exclude honest mixes. However, we find it useful to maintain *thresh* as a separate value, to allow the use of larger value for *thresh* to account for a number of failures of honest mixes or links between honest mixes, when the Miranda design is adopted by a practical system.

Significant, although not prohibitive, processing and communication is involved in the inter-epoch process; this motivates the use of longer epochs. On the other hand, during an entire epoch, we use a fixed set of cascades, which may reduce due to failures; and clients may not be fully aware of links and mixes detected as faulty. This motivates the use of shorter epochs. These considerations would be balanced by the designers of an anonymous communication system, as they incorporate the Miranda design.

4 Intra-Epoch Process

In this section, we present the mechanisms that operate during an epoch to deter active attacks, including dropping attacks. We start by describing how active attacks are detected and how this deters malicious behavior. Next, we discuss nodes who refuse to cooperate.

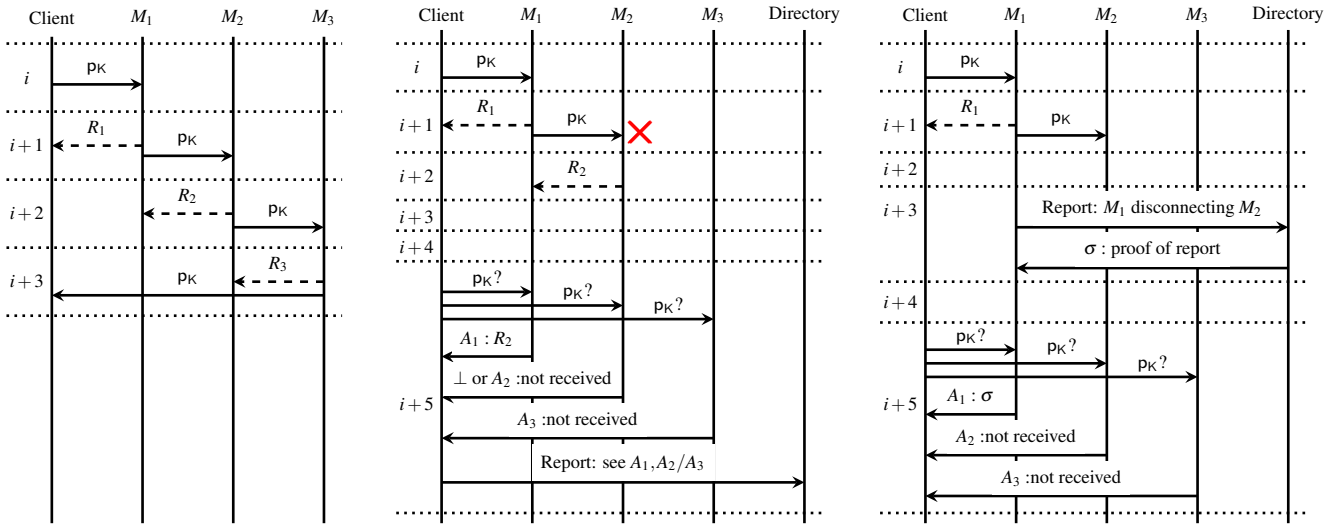
Note that in this section, as in the entire Miranda design, we assume reliable communication between any pair of honest participants. As we explain in Subsection 2.2, a practical system deploying Miranda should use a lower-layer protocol to deal with (even severe) packet losses, and we developed such efficient protocol - see [5].

4.1 Message Sending

At the beginning of each epoch, clients acquire the list of all currently available cascades from the directory authorities. When Alice wants to send a message, her client filters out all cascades containing mixes through which she does not wish to relay messages. We denote the set of cascades selected by Alice as C_A. Next, Alice picks a random cascade from C_A, which she uses throughout the whole epoch, and encapsulates the message into the packet format. For each mix in the cascade, we include in the routing information the exact round number during which the mix should receive the packet and during which it should forward it. Next, the client sends the encoded packet to the first mix on the cascade. In return, the mix sends back a *receipt*, acknowledging the received packet.

4.2 Processing of Received Packets

After receiving a packet, the mix decodes a successive layer of encoding and verifies the validity of the expected round r and well-formedness of the packet. At the end of the round, the mix forwards all valid packets to their next hops. Miranda requires mixes to acknowledge received packets by sending back receipts. A receipt is a digitally signed [31]



(a) Successful loop packet p_k sent during round i and received during round $i+3$. Each mix M_i sends back receipt R_i.

(b) Example of naive dropping of loop message p_k by M_2, which drops p_k yet sends back a receipt. Since p_k did not come back the client queries all mixes during round $i+5$ for the proof of forwarding. M_2 either claims that it did not receive p_k (A_2), thus providing the client a proof that conflicts with the receipt R_2, or M_2 does not cooperate ($\perp$). In both cases the directory authority verifies received A_i's and excludes malicious M_2.

(c) Loop packet fails to complete the loop due to non-responding mix. M_1 did not receive receipt from M_2 on round $i+2$ and issues a disconnection in round $i+3$. The client performs the query phase on round $i+5$ and receives the proof of disconnection. The result: M_2 failed to send a receipt to M_1, and thus lost the link to it.

Figure 2: A diagram illustrating loop packets and isolation process. We denote receipt from mix M_i as R_i, and the response as A_i. Note that both in (b) and (c) the entire query and report phases occur during round $i+5$, but it could also be spanned across several rounds, as long as it has a bounded time-frame. For example, if desired, answering the query for p_k could be done in round $i+6$ instead of limiting it to the same round.

statement confirming that a packet p was received by mix M_i. Receipts must be sent and received by the preceding mix within the same round in which packet p was sent.

Generating receipts. For simplicity, we denote a receipt for a single packet p as $\text{receipt} \leftarrow \text{Sign}(\text{p} \,||\, \text{receivedFlag} = 1)$, where $\text{Sign}(\cdot)$ is a secure digital signature algorithm, and $\text{Verify}(\cdot)$ is its matching verification function[3]. However, generating receipts for each packet individually incurs a high computational overhead due to costly public key signature and verification operations.

To reduce this overhead, mixes gather all the packets they received during round r in Merkle trees [37] and sign the root of the tree once. Clients' packets are grouped in a single Merkle tree T_C and packets from mix M_i are grouped in a Merkle tree T_{M_i}. Mixes then generate two types of receipts: (1) receipts for clients and (2) aggregated receipts for mixes. Each client receives a receipt for each message she sends. Client receipts are of the form: $\text{receipt} = (\sigma_\text{C}, \Gamma_\text{p}, \text{r})$, where: σ_C is the signed root of T_C, Γ_p is the appropriate information needed to verify that packet p appears in T_C, and r is the round number. Similarly, each mix, except the last one, receives a receipt in response to all the packets it forwarded in the last round. However, unlike client receipts, mixes expect back a *single* aggregated receipt for all the packets they sent to a specific mix. An aggregated receipt is in the form of: $\text{receipt} = (\sigma_i, \text{r})$, where: σ_i denotes the signed root of T_{M_i} and r is the round number. Since mixes know which packets they forwarded to a particular mix, they can recreate the Merkle tree and verify the correctness of the signed tree root using a single receipt. Once a mix sent an aggregated receipt, it expects back a signed confirmation on that aggregated receipt, attesting that it was delivered correctly. Mixes record the receipts and confirmations to prove later that they behaved honestly in the mixing operation.

Lack of a receipt. If a mix does not receive an aggregated receipt or does not receive a signed confirmation on an aggregated receipt it sent within the expected time slot[4], the mix disconnects from the misbehaving mix. The honest mix detaches from the faulty mix by informing the directory authorities about the disconnection through a *signed link disconnection receipt*. Note, that the directories cannot identify which of the disconnecting mixes is the faulty one merely based on this message, because the mix who sent the complaint might be the faulty one trying to discredit the hon-

[3] Although Sign and Verify use the relevant cryptographic keys, we abuse notations and for simplicity write them without the keys.

[4] Recall that we operate in a synchronous setting, where we can bound the delay of an acknowledgement.

est one. Therefore, the directory authorities only disconnect the ***link*** between the two mixes. The idea of disconnecting links was earlier investigated in various Byzantine agreement works [23], however, to our knowledge this approach was not yet applied to the problem of mix network reliability.

Anonymity loves company. Note, however, that this design may fail even against an attacker who does not control any mix, if a cascade receives less than the *minimal anonymity set size* ω. We could ignore this as a very unlikely event, however, Miranda ensures anonymity also in this case - when the first mix is honest. Namely, if the first mix receives less than ω messages in a round, it would not forward any of them and respond with a special 'under-ω receipt' explaining this failure. To prevent potential abuse of this mechanism by a corrupt first mix, which receives over ω messages yet responds with under-ω receipt, these receipts are shared with the directories, allowing them to detect such attacks.

4.3 Loop Messages: Detect Stealthy Attacks

In a *stealthy active attack*, a mix drops a message - yet sends a receipt as if it forwarded the message. To deter such attacks, clients periodically, yet randomly, *send loop messages to themselves*. In order to construct a *loop message*, the sender S, chooses a unique random bit-string K_S. Loop messages are encoded in the same manner as regular messages and sent through the same cascade C selected for the epoch, making them *indistinguishable* from other messages at any stage of their routing. The *loop message* is encapsulated into the packet format as follows:

$$\mathsf{p_K} \leftarrow \mathsf{Pack}(\mathsf{path}=\mathsf{C}, \mathsf{routingInfo}=\mathsf{routing}, \mathsf{rnd}=\mathsf{H}(\mathsf{K_S}) \\ \mathsf{recipient}=\mathsf{S}, \mathsf{message}=\text{“loop”})$$

The tuple $(\mathsf{S}, \mathsf{K_S}, \mathsf{C}, \mathsf{routing})$ acts as the *opening value*, which allows recomputing $\mathsf{p_K}$ as well as all its intermediate states $\mathsf{p_K^i}$ that mix M_i should receive and emit. Therefore, revealing the *opening value* convinces everyone that a particular packet was indeed a *loop message* and that its integrity was preserved throughout its processing by all mixes. Moreover, the construction of the *opening value* ensures that only the creator of the loop packet can provide a valid *opening value*, and no third party can forge one. Similarly, nobody can reproduce an opening value that is valid for a non-loop packet created by an honest sender.

If a loop message fails to complete the loop back, this means that one of the cascade's mixes misbehaved. The sender S queries all the mixes in the cascade for evidence whether they have received, processed and forwarded the loop packet. This allows S to isolate the cascade's problematic link or misbehaving mix which caused the packet to be dropped. S then reports the isolated link or mix to the directory authorities and receives a signed confirmation on her report. This confirmation states that the link will no longer be used to construct future cascades. We detail the querying and isolation process in Section 4.3.2.

4.3.1 When to send loop messages?

The sending of loop messages is determined according to α, which is the required *expected probability of detection* - a parameter to be decided by the system designers. Namely, for every message, there is a fraction α chance of it being a loop message. To achieve that, if Alice sends β messages in round r, then $\lceil \frac{\alpha \cdot \beta}{1-\alpha} \rceil$ additional loop messages are sent alongside the genuine messages.

This may seem to only ensure α in the context of the messages that Alice *sends* but not against an attack on messages *sent to* Alice. However, notice that if a corrupt mix M_i drops messages sent to Alice by an honest sender Bob, then M_i faces the same risk of detection - by Bob.

If Alice can sample and estimate an upper bound γ on the number of messages that she will *receive* in a particular round, then she can apply additional defense. Let x be the number of rounds that it takes for a loop message to come back, and let r denote the current round. Let's assume that Alice knows bound γ on the maximal number of messages from honest senders, that she will receive in round $\mathsf{r}+\mathsf{x}$. Then, to detect a mix dropping messages sent to her with probability α, it suffices for Alice to send $\lceil \frac{\alpha \cdot \gamma}{1-\alpha} \rceil$ loop messages in round r. More precisely, given that Alice sends β messages in round r, in order for the loop messages to protect both messages sent in that round and messages received in round $\mathsf{r}+\mathsf{x}$ she should send $\lceil \frac{\alpha \cdot \max(\beta,\gamma)}{1-\alpha} \rceil$ loop messages in round r.

Within-round timing. If the Miranda senders would send each message immediately after receiving the message from the application, this may allow a corrupt first mix to distinguish between a loop message and a 'regular' message. Namely, this would occur if the attacker knows the exact time at which the application calls the 'send' function of Miranda to send the message. To foil this threat, in Miranda, messages are always sent only during the round following their receipt from the application, and after being shuffled with all the other messages to be sent during this round.

4.3.2 Isolating corrupt mixes with loop messages

Since clients are both the generators and recipients of the attack-detecting *loop messages*, they know exactly during which round r the loop should arrive back. Therefore, if a *loop message* fails to complete the loop back to the sender as expected, the client initiates an *isolation* process, during which it detects and isolates the specific problematic node or link in the cascade. The isolation process starts with the client querying each of the mixes on the cascade to establish whether they received and correctly forwarded the loop packet. During the querying phase, the client first reveals to the respective mixes the packet's *opening value*, in order

to prove that it was indeed a loop packet. Next, the client queries the mixes for the receipts they received after they delivered that packet. When clients detect a problematic link or the misbehaving mix, they report it to the directory authorities, along with the necessary proofs that support its claim. This is in fact a broadcasting task in the context of the well-known *reliable broadcast problem* and can be solved accordingly [36]. Each directory authority that receives the report verifies its validity, and if it is correct, stores the information to be used in future cascade generation processes. Then, the client chooses another cascade from the set of available cascades and sends future packets and loop messages using the new route. For an illustration of loop packets and the isolation process, see Figure 2.

When a client asks an honest mix to prove that it received and correctly forwarded a packet, the mix presents the relevant receipt. However, if a mix did not receive this packet, it attests to that by returning an appropriate signed response to the client. If a loop message did not complete the loop because a malicious mix dropped it and did not send a receipt back, the honest preceding mix would have already disconnected from the misbehaving mix. Thus, the honest mix can present the appropriate *disconnection receipt* it received from the directory authorities as an explanation for why the message was not forwarded (see Figure 2c).

The malicious mix can attempt the following actions, in order to perform an active attack.

Naive dropping. A mix which simply drops a loop packet after sending a receipt to the previous mix can be detected as malicious beyond doubt. When the client that originated the dropped loop packet queries the preceding mix, it presents the receipt received from the malicious mix, proving that the packet was delivered correctly to the malicious node. However, the malicious mix is unable to produce a similar receipt, showing that the packet was received by the subsequent mix, *or* a receipt from the directories proving that it reported disconnection from the subsequent mix. The malicious mix may simply not respond at all to the query. However, the client will still report to the directories, along with the proofs from the previous and following mixes, allowing the directories to resolve the incident (contacting the suspected mix themselves to avoid any possible 'framing') (see Figure 2b).

Blaming the neighbors. Malicious mixes performing active dropping attacks would prefer to avoid complete exclusion. One option is to drop the packet, and not send a receipt to the previous mix. However, this causes the preceding mix to disconnect from the malicious one at the end.

Alternatively, the corrupt mix may drop the packet after it generates an appropriate receipt. To avoid the risk of its detection as a corrupt mix, which would happen if it was a loop message, the corrupt mix may disconnect from the subsequent mix - again losing a link. Therefore, a corrupt mix that drops a packet either loses a link, or risks being exposed (by loop message) and removed from the network.

Delaying packets. A malicious mix can also delay a packet instead of dropping it, so that the honest subsequent mix will drop that packet. However, the honest subsequent mix still sends a receipt back for that packet, which the malicious mix should acknowledge. If the malicious mix acknowledges the receipt, the malicious mix is exposed when the client performs the *isolation process*. The client can obtain a signed receipt proving that the malicious mix received the packet on time, and also the acknowledged receipt from the honest mix that dropped the delayed packet. The latter contains the round number when the packet was dropped, which proves the malicious mix delayed the packet and therefore should be excluded. Otherwise, if the malicious mix refuses to sign the receipt, the honest mix disconnects from the malicious one. Therefore, the delaying attack also causes the mix to either lose a link or to be expelled from the system.

The combination of packet receipts, link disconnection notices, the isolation process and loop messages, forces malicious mixes to immediately lose links when they perform active attacks. Failure to respond to the preceding mix or to record a disconnection notice about the subsequent mix in a timely manner creates potentially incriminating evidence, that would lead to a complete exclusion of the mix from the system. This prevents malicious mixes from silently attacking the system and blaming honest mixes when they are queried in the isolation mechanism. The mere threat of loop messages forces malicious mixes to drop a link with an honest mix for each message they wish to suppress, or risk exposure.

4.4 Handling missing receipts

Malicious mixes might attempt to circumvent the protocol by refusing to cooperate in the isolation procedure. Potentially, this could prevent clients from obtaining the necessary proofs about problematic links, thus preventing them from convincing directory authorities about problematic links. If malicious mixes refuse to cooperate, clients contact a directory authority and ask it to perform the isolation process on their behalf. Clients can prove to the directory authorities that the loop packet was indeed sent to the cascade using the receipt from the first mix. If all mixes cooperate with the directory authority, it is able to isolate and disconnect the problematic link. Otherwise, if malicious mixes do not cooperate with the directory authority, it excludes those mixes from the system.

We note that a malicious client may trick the directory authorities into performing the isolation process on its behalf repeatedly, against honest mixes. In that case, directory authorities conclude that the mix is honest, since the mix can provide either a receipt for the message forwarded or a disconnection notice. However, this is wasteful for both direc-

tory authorities and mixes. Since clients do not have to be anonymous vis-a-vis directory authorities, they may record false reports and eventually exclude abusive clients. Furthermore, the clients have to send proofs from the following mix of not having received the packet, which cannot be done if there was no mix failure.

Malicious entry mix. If a first mix does not send a receipt, the client could have simply chosen another cascade; however, this allows malicious mixes to divert traffic from cascades which are not fully malicious, without being penalized, increasing the probability that clients would select other fully malicious cascades instead. To avoid that, in Miranda, clients *force* the first mix to provide a receipt, by relaying the packet via a trusted ***witness***. A witness is just another mix that relays the packet to the misbehaving first mix. Now, the misbehaving node can no longer refuse to produce a receipt, because the packet arrives from a mix, which allows the isolation process to take place. Note that since a witness sends messages on behalf of clients, the witness *relays* messages without the ω constraint (as if it was a client).

If the witness itself is malicious, it may also refuse to produce a receipt (otherwise, it loses a link). In that case, the client can simply choose another witness; in fact, if desired, clients can even send via multiple witnesses concurrently to reduce this risk - the entry mix can easily detect the 'duplicate' and handle only one message. This prevents malicious mixes from excluding semi-honest cascades without losing a link. Moreover, although the refused clients cannot prove to others that they were rejected, they can learn about malicious mixes and can avoid all future cascades that contain them, including fully malicious cascades, which makes such attacks imprudent.

5 Inter-Epoch Process

In this section, we discuss the inter-epoch operations, taking place toward the end of an epoch; upon its termination, we move to a new epoch. The inter-epoch process selects a new random set of cascades to be used in the coming epoch, avoiding the links reported by the mixes, as well as any mixes detected as corrupt.

Until the inter-epoch terminates and the mixes move to the new epoch, the mixes continue with the intra-epoch process as before; the only difference is that newly detected failures, would be 'buffered' and handled only in the following run of the inter-epoch process, to avoid changing the inputs to the inter-epoch process after it has begun.

The inter-epoch process consists of the following steps.

5.1 Filtering Faulty Mixes

Directory authorities share amongst themselves the evidences they received and use them to agree on the set of faulty links and mixes. The evidences consist of the reports of faulty links from mixes, clients or authorities performing the *isolation process*. The directory authorities exchange all new evidences of faulty links and mixes, i.e., not yet considered in the previous inter-epoch computation process. Every directory can validate each evidence it received and broadcast it to all other directories. Since we assume majority of honest directories and synchronous operation, we can use known broadcast/consensus protocols, and after a small number of rounds, all honest directory authorities have exactly the same set of faulty links.

Note, that only links connected to (one or two) faulty mixes are ever disconnected. Hence, any mix which has more than $thresh$ links disconnected must be faulty (due to the assumption that $thresh > n_m$), and hence the directories exclude that mix completely and immediately. Since the directory authorities share exactly the same set of faulty links, it follows that they also agree on exactly the same set of faulty mixes. We call this exclusion process a *simple malicious mix filtering* step. In Section 6, we discuss more advanced filtering techniques, based on *community detection*.

Simple malicious mix filtering technique. To perform the simple malicious mix filtering, each directory authority can build a graph that represents the connectivity between mixes. Namely, consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertices map to the mixes in the system ($\mathcal{V} = \mathcal{M}$), and an edge $(M_i, M_j) \in \mathcal{E}$ means that the link between mixes M_i and M_j was not dropped by either mix. Let $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$ be the complement graph of $\mathcal{G}$ and let $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i)$ denote the degree of the vertex M_i in graph $\overline{\mathcal{G}}$. In the beginning, before any reports of faults have arrived at the directory authorities, $\mathcal{G}$ is a complete graph and $\overline{\mathcal{G}}$ is an empty graph. As time goes by, $\mathcal{G}$ becomes sparser as a result of the links being dropped, and proportionally, $\overline{\mathcal{G}}$ becomes more dense. The filtering mechanism removes all mixes that lost $thresh$ links or more, i.e., $\{M_i \mid \forall M_i \in \overline{\mathcal{G}} : \ \mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) \geq thresh\}$, where $thresh = n_m + 1$. The filtering mechanism checks the degree $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i)$ in graph $\overline{\mathcal{G}}$, since the degree in $\overline{\mathcal{G}}$ represents how many links M_i lost. We emphasize that when such malicious mix is detected and removed, the number of malicious mixes in the system is decreased by one ($n_m = n_m - 1$) and proportionally so does $thresh$ ($thresh = thresh - 1$). As a result, whenever the mechanism removes a malicious mix it repeats the mechanism once again, to see whether new malicious mixes can be detected according to the new $thresh$ value. An illustration of this process is depicted in Figure 3.

5.2 Cascades Selection Protocol

After all directory authorities have the same view of the mixes and their links, they select and publish a (single) set of cascades, to be used by all clients during the coming epoch. To allow clients to easily confirm that they use the correct set of cascades, the directory authorities collectively sign the set

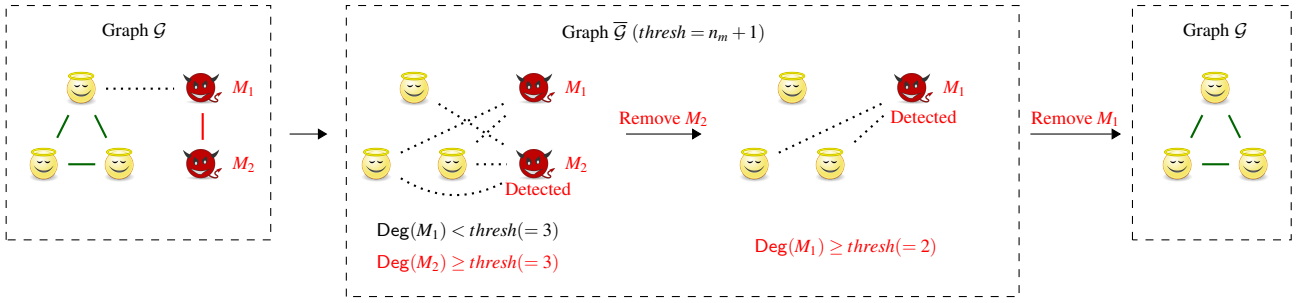


Figure 3: An illustration of the simple malicious mix filtering (without community detection).

that they determined for each epoch, using a threshold signature scheme [46, 25]. Hence, each client can simply retrieve the set from any directory authority and validate that it is the correct set (using a single signature-validation operation).

The *cascades selection protocol* allows all directory authorities to agree on a random set of cascades for the upcoming epoch. The input to this protocol, for each directory authority, includes the set of mixes $\mathcal{M}$, the desired number of cascades to be generated $\mathsf{n_c}$, the length of cascades ℓ and the set of faulty links $\mathcal{F}_\mathcal{L} \subset \mathcal{M} \times \mathcal{M}$. For simplicity, $\mathcal{M}$, $\mathsf{n_c}$ and ℓ are fixed throughout the execution.

The goal of all directory authorities is to select the same set of cascades $\mathcal{C} \subseteq \mathcal{M}^\ell$, where $\mathcal{C}$ is uniformly chosen from all sets of cascades of length ℓ, limited to those which satisfy the selected *legitimate cascade predicates*, which define a set of constraints for building a cascade. In [34], we describe several possible legitimate cascade predicates, and discuss their differences.

Given a specific legitimate cascade predicate, the protocol selects the same set of cascades for all directory authorities, chosen uniformly at random among all cascades satisfying this predicate. This is somewhat challenging, since sampling is normally a random process, which is unlikely to result in exactly the same results in all directory authorities. One way of ensuring correct sampling and the same output, is for the set of directories to compute the sampling process jointly, using a multi-party secure function evaluation process, e.g., [26]. However, this is a computationally-expensive process, and therefore, we present a much more efficient alternative. Specifically, all directories run exactly the same sampling algorithm and for each sampled cascade validate it using exactly the same legitimate cascade predicate. To ensure that the results obtained by all honest directory authorities are identical, it remains to ensure that they use the same random bits as the seed of the algorithm. To achieve this, while preventing the faulty directory authorities from biasing the choice of the seed bits, we can use a coin-tossing protocol, e.g., [7], among the directory authorities[5].

[5]Note, that we only need to generate a small number of bits (security parameter), from which we can generate as many bits as necessary using a pseudo-random generator.

6 Community-based Attacker Detection

So far, the discussion focused on the core behaviour of Miranda and presented what Miranda can do and how it is done. Interestingly, Miranda's mechanisms open a doorway for advanced techniques, which can significantly improve the detection of malicious mixes. In this section, we discuss several techniques that can leverage Miranda's faulty links identification into a powerful tool against malicious adversaries. Among others, we use community detection techniques. Community detection has been used in previous works to achieve Sybil detection based on social or introduction graphs [16, 17]. However, we assume that the problem of Sybil attacks is solved through other means, such as admission control or resource constraints. Encouragingly, many other techniques can be employed; yet, we hope that the following algorithms will be also useful in other applications where applicable, e.g., where community detection is needed.

We begin with the following observation.

Observation 1. *For every two mixes M_i, M_j that have an edge in $(M_i, M_j) \in \overline{\mathcal{E}}$, at least one of them is a malicious mix.*

Observation 1 stems directly from our assumption that honest mixes never fail. Therefore, a dropped link must be between either an honest mix and a malicious mix or between two malicious mixes. Following this observation, one possible strategy is *aggressive pair removal*, i.e., remove both mixes, if one or both of them report failure of the link connecting them. This strategy seems to provide some benefits - the adversary seems to 'lose more', however it comes at an excess cost of possible exclusion of honest nodes. Therefore, we focus on less aggressive techniques that exclude malicious mixes *without* excluding also honest ones.

Threshold Detection Algorithm. Since the *aggressive removal* of both mixes connected by the failed link from $\mathcal{G}$ is not efficient, we adopt the idea of *virtual removal* of the conflicting pair. By *virtually* we mean that virtually removed mixes are not classified as malicious and they are only removed from $\overline{\mathcal{G}}$ for the duration of the algorithm's execution, and not from $\mathcal{G}$ nor $\mathcal{M}$. We present the *Threshold Detec-*

(a) Graph $\overline{\mathcal{G}}$, the simple malicious mix filtering technique cannot detect M_2 because $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 < 3 = thresh$.

(b) An execution of the *ThresholdDetection* on the same $\overline{\mathcal{G}}$ graph virtually removes M_1 and M_5 which decreases *thresh*, resulting in $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 \geq 2 = thresh$ and detection of M_2 as a malicious mix.

Figure 4: An illustration of how virtually removing mixes from $\overline{\mathcal{G}}$ can expose malicious mixes. Algorithm 2 refers to the graph in 4b as $\overline{\mathcal{G}_1}$, since it is the same graph $\overline{\mathcal{G}}$ as in 4a but without M_1 and without M_1's neighbors.

tion technique in Algorithm 1. The algorithm takes as input graph $\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}})$, where an edge $(M_i, M_j) \in \overline{\mathcal{E}}$ represents the disconnected link between M_i and M_j. The algorithm starts by invoking the SIMPLEMALICIOUSFILTERING procedure (described in Section 5.1) on the graph $\overline{\mathcal{G}}$ (line 12). Next, the algorithm invokes the VIRTUALPAIRREMOVAL procedure on $\overline{\mathcal{G}}$ to *virtually* remove a pair of mixes from $\overline{\mathcal{G}}$ (line 14). Following observation 1, at least one malicious mix was *virtually* removed, thus the *virtual* threshold $thresh'$ value is decreased by 1 (line 15). We use the $thresh'$ variable to keep track of the virtually removed malicious mixes and the global *thresh* value is decreased only when a malicious mix was actually detected (line 4), and the rest only change the virtual threshold $thresh'$. After that, the algorithm invokes the procedure SIMPLEMALICIOUSFILTERING again on the *updated* $\overline{\mathcal{G}}$ graph, i.e., without the pair of mixes that were virtually removed by the VIRTUALPAIRREMOVAL procedure. The algorithm repeats lines 14-16 as long as there are edges in $\overline{\mathcal{G}}$. For an illustration why the *ThresholdDetection* algorithm is better than the original *simple malicious mix filtering* see Figure 4.

We next improve upon the detection of malicious mixes by the *ThresholdDetection* algorithm, while still never removing honest mixes. Our improvement is based on Observation 2 below; but before presenting it, we need some preliminaries.

We first define a simple notion which can be applied to any undirected graph. Specifically, let $G^0 = (V^0, E^0)$ be an arbitrary undirected graph. A sequence $\{G^j\}_{j=0}^{\mu}$ of subgraphs of G^0 is a *removal sequence* of length $\mu \geq 1$ of G^0, if for every $j : \mu \geq j \geq 1$, $G^j = G^{j-1} - v_j$. Namely, G^j is the same as G^{j-1}, except for removal of some node $v_j \in G^{j-1}$, and of all edges connected to v_j. A removal sequence is *legitimate* if every removed node v_j has at least one edge.

Let us define the graph $\overline{\mathcal{G}_i}$ to be the resulting graph after removing from $\overline{\mathcal{G}}$ the node M_i together with all its *neighbors*, denoted as $\mathsf{N}(M_i)$.

Algorithm 1 *ThresholdDetection*$(\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}}))$

```
1:  procedure SIMPLEMALICIOUSFILTERING(G, thresh')
2:      for every M_i ∈ G s.t. Deg_G(M_i) ≥ thresh' do
3:          M_i is malicious (remove from G, G, M).
4:          thresh ← thresh − 1
5:          thresh' ← thresh' − 1
6:
7:  procedure VIRTUALPAIRREMOVAL(G)
8:      Pick an edge (M_i, M_j) ∈ E.
9:      Remove mixes M_i, M_j from G.
10:
11: thresh' ← thresh.
12: Invoke SIMPLEMALICIOUSFILTERING(G).
13: while E ≠ ∅ do
14:     Invoke VIRTUALPAIRREMOVAL(G).
15:     thresh' ← thresh' − 1.
16:     Invoke SIMPLEMALICIOUSFILTERING(G).
```

Observation 2. *If $\overline{\mathcal{G}_i}$ has a legitimate removal sequence of length μ_i, then there are at least μ_i malicious nodes in $\overline{\mathcal{G}_i}$.*

We use Observation 2 to identify malicious mixes, using the following claim.

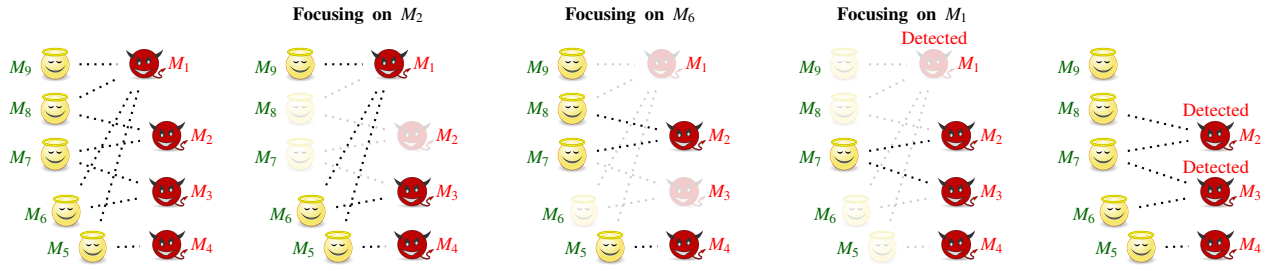
Claim 1. *Every node M_i that satisfies $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) > n_m - \mu_i$ is a malicious node.*

Proof. Assume to the contrary, that there exists a mix M_i such that $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) > n_m - \mu_i$ but M_i is an honest mix. Since there are n_m malicious mixes in $\mathcal{M}$, and μ_i of them are not neighbors of M_i, then the maximum number of malicious mixes that can be also neighbors of M_i is $n_m - \mu_i$, since M_i is honest. But if $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) > n_m - \mu_i$, then at least one of the neighbors of M_i is also honest, which contradicts the assumption that honest links never fail. Therefore, if $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) > n_m - \mu_i$ then M_i must be a malicious mix. □

For example, see Figure 4b which depicts the graph $\overline{\mathcal{G}_1}$. By observing $\overline{\mathcal{G}_1}$, we know that at least one of the mixes M_2, M_3 are malicious (since they share an edge), therefore, $\mu_i \geq 1$ since we successfully identified a malicious mix which is not in $\{M_1 \cup \mathsf{N}(M_1)\}$. Alternatively, the same argument can be made regarding M_2 and M_4 *instead* of the pair M_2 and M_3. Since after removing M_2, M_4 from $\overline{\mathcal{G}_1}$ there are no edges left in $\overline{\mathcal{G}_1}$, then $\mu_1 = 1$.

Algorithm 2 presents the *Community Detection* algorithm, which leverages Claim 1 to detect malicious mixes. An illustration of the operation of this algorithm is demonstrated in Figure 5.

Notice that the algorithm only examines nodes with a degree larger than 1 (line 3). The reason is that if $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) = 0$ then M_i did not perform an active attack yet, thus it cannot be detected, and if $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) = 1$ then M_i cannot be classified based on its neighbors. Therefore, an execution of the



(a) $\forall M_i : \mathrm{Deg}_{\overline{\mathcal{G}}}(M_i) < thresh$, simple malicious mix filtering does not detect malicious mixes.

(b) When we observe $\overline{\mathcal{G}_2}$, i.e., $\overline{\mathcal{G}}$ after the removal of M_2 and $N(M_2)$, two scenarios are possible. In the first scenario, $\mu_2 = 3$ (e.g., if M_1 and M_9 are removed first), thus $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 > 1 = n_m - \mu_2$, and therefore M_2 is detected as malicious. In the second scenario, $\mu_2 = 2$ (e.g., if M_1 and M_6 are removed first), thus $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 \leq 2 = n_m - \mu_2$, and therefore M_2 is *not* detected as malicious (yet). A similar situation occurs with M_3 when observing $\overline{\mathcal{G}_3}$.

(c) When we observe $\overline{\mathcal{G}_6}$, two malicious mixes can be identified, thus $\mu_6 = 2$. As a result, since $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_6) = 2 \leq 2 = n_m - \mu_6$, M_6 is not classified as malicious (nor should it be). Note that even if M_3 was removed in (b), then $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_6) = 1$ and therefore the algorithm cannot classify it based on its neighbors. The same explanations apply to the rest of the honest mixes.

(d) When we observe $\overline{\mathcal{G}_1}$, only one malicious mix can be identified, thus $\mu_1 = 1$. As a result, since $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_1) = 4$ is larger than $n_m - \mu_1 = 3$, M_1 is detected as malicious.

(e) If M_2 and M_3 were not detected as malicious as explained in (b), then after the removal of M_1 in (c) they will be detected, because the removal of M_1 causes $n_m = 4 \rightarrow n_m = 3$. Since the algorithm runs in a loop, when the algorithm will re-check $\overline{\mathcal{G}_2}$, it will discover that $\mu_2 = 2$ and thus $\mathrm{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 > 1 = n_m - \mu_1$, which results in removal of M_2. The same goes for M_3. After the removal of M_1, M_2 and M_3, the algorithm cannot classify M_4 as malicious based on its neighbors, since M_4 only dropped one link. However, the algorithm has the option to *aggressively* remove both M_4, M_5.

Figure 5: A demonstration how Miranda's community detection can significantly improve the detection of malicious mixes using an example graph $\overline{\mathcal{G}}$ and $thresh = n_m + 1$.

Algorithm 2 *CommunityDetection*$(\overline{\mathcal{G}} = (\overline{\mathcal{V}}, \overline{\mathcal{E}}))$

```
1:  n'_m ← n_m
2:  while Ē ≠ ∅ do
3:      for each M_i ∈ V̄ s.t. Deg_Ḡ(M_i) > 1 do
4:          Construct Ḡ_i = (V̄_i, Ē_i) from Ḡ.
5:          μ_i ← 0
6:          while Ē_i ≠ ∅ do
7:              Invoke VirtualPairRemoval(Ḡ_i).
8:              μ_i ← μ_i + 1
9:          if Deg_Ḡ(M_i) > n'_m − μ_i then
10:             M_i is malicious (remove from G, Ḡ, M).
11:             n_m ← n_m − 1, n'_m ← n'_m − 1
12:     if Ē ≠ ∅ then
13:         Invoke VirtualPairRemoval(Ḡ).
14:         n'_m ← n'_m − 1
```

CommunityDetection might not be able to detect all malicious mixes that exposed themselves, e.g., mixes with a degree that equals to 1. If desired, there is always the opportunity to execute the aggressive pair removal technique *after* the *CommunityDetection* algorithm to potentially remove more malicious mixes (with price of possible removal of an honest mix). Also, randomly picking a pair of mixes that share an edge in $\overline{\mathcal{G}}$ might not always be the optimal strategy. In small graphs, the algorithm can exhaust all possible removal variations, but this is a time-consuming option in large graphs. A more sophisticated picking strategy might yield better results; however, when we experimented with some possible strategies, we did not notice a significant improvement over the random picking strategy.

The techniques discussed in this section provide Miranda a significant advantage, since malicious mixes can be detected even if they do not pass *thresh*. Merely the threat of such techniques is significant in deterring active attacks. In Section 7.4 we analyze the security of the mechanisms discussed here and evaluate them empirically. In [34] we present alternative scheme for community detection based on random walks.

7 Analysis of Active Attacks

In this section, we analyze the impact of active attacks in the presence of Miranda. We first analyze Miranda against traditional and non-traditional active attacks, including attacks designed to abuse the protocol to increase the chances of clients choosing fully malicious cascades. We continue by examining the security of loop messages and conclude this section by evaluating how community detection strengthens Miranda.

7.1 Resisting Active Attacks

As discussed in Section 4, a malicious mix that drops a packet sent from a preceding mix or destined to a subsequent mix, loses at least one link; in some cases, the malicious mix gets completely excluded. Hence, the adversary quickly loses its attacking capabilities, before any significant impact is introduced. However, the adversary might try other approaches in order to link the communicating users or gain advantage in the network, as we now discuss.

A malicious first mix can refuse clients' packets; however, such attack is imprudent, since clients can migrate to other cascades. Furthermore, clients can force the malicious mix to relay their packets, using a witness. Similarly, it is ineffective for the last mix of a cascade to drop *all* packets it receives, since clients learn through isolation that the dropped loop packets successfully arrived at the last mix. Although clients cannot prove the mix maliciousness, they avoid future cascades containing the malicious mix, including fully malicious cascades.

Instead of directly dropping packets, adversaries can cause a packet to be dropped by delaying the packet. However, such attack is also detected.

Claim 2. *A malicious mix that delays a packet, is either expelled from the system or loses a link.*

Argument. When an honest mix receives a delayed packet, it drops it. However, the honest mix still sends a receipt back for that packet. If the malicious mix acknowledges the receipt, the malicious mix is exposed when the client performs the isolation process: the client can obtain a signed receipt proving that the malicious mix received the packet on time, and also the acknowledged receipt from the honest mix that dropped the delayed packet. The latter contains the round number when the packet was dropped, which proves the malicious mix delayed the packet and therefore should be excluded. Otherwise, if the malicious mix refuses to sign the receipt, the honest mix disconnects from the malicious mix. □

Injecting malformed packets. Notice how the honest mix that dropped the delayed message still sends back a receipt for it. The reason is that the dropping mix cannot be sure that the previous mix did delay the message. Instead, this can be the result of an adversary that crafts a packet with the same round number in two successive layers.

Claim 3. *An adversary cannot craft a loop message that causes a link loss between two honest mixes.*

Argument. Any loop message has to be well-formed in order for directory authorities to accept it. An adversary can craft a message with invalid round numbers in the packet's routing information, which would cause the honest mix to drop the packet. However, although the honest mix drops the packet, it still sends back a receipt for that packet. Otherwise, the preceding mix, which has no way of knowing that the next layer is intentionally malformed, would disconnect from the subsequent mix. While the adversary can obtain a proof showing that a loop message was dropped, it cannot prove that the loop message was well-formed. □

Aggressive active attacks. In order to de-anonymize the network users, the adversary can choose a more aggressive approach and drop a significant number of packets. For example, in the $(n-1)$ attack [45] applied to the full network, the adversary tracks a target packet from Alice by blocking other packets from arriving to an honest mix, and instead injecting their own packets. Another example is the intersection attack [8], where the adversary tries disconnecting target clients. If the adversary cannot directly disconnect a client with a targeted attack, it can disconnect a client by dropping an entire batch of packets where one of them belongs to the client (the adversary simply does not know which). However, it is important to note, that if an adversary can engineer a scenario where a single target packet is injected and mixed with only messages that the adversary controls, *any* mix-based system is vulnerable. Nevertheless, we argue that Miranda inflicts serious penalty on the adversary who attempts to perform an aggressive dropping of packets.

Claim 4. *Miranda deters aggressive active attacks.*

Argument. Aggressive active attacks require the ability to drop many packets. In Miranda, a malicious mix that drops any packet from another mix without sending back a receipt, loses a link (see Section 4 and Figure 2c). Alternatively, if the malicious mix drops packets but does send receipts for these dropped packets, clients can prove that the malicious mix received their (loop) packets and did not forward them, which results in the exclusion of the malicious mix (see Figure 2b). A malicious entry mix may drop packets from clients, since losing a link to a client is not a serious 'penalty'; but in Miranda, clients then use a *witness mix* (see Section 4.4) – forcing the mix to either relay their packets, or - lose a link to a mix or risk discovery, as discussed above.

Miranda enforces a minimum number of ω packets for mixing by the entry mix. This is designed to protect the *rare* cases where a client sends via an entry mix which is used only by few (or no) other clients, which could allow an eavesdropper attack; we now explain why this cannot be abused to facilitate an active attack (by the first mix).

Recall, that in this case, as in our entire analysis of corrupt-mix attacks, we assume that at least 2ω honest clients send packets to the (possibly corrupt) entry mix; and, as mentioned above, the mix cannot simply 'drop' these (since clients will use witness and then the corrupt mix will lose - at least - a link).

Instead, the corrupt mix could send to these clients, or most of them, the special 'under-ω receipt', claiming (incorrectly) that it didn't receive ω messages during this round.

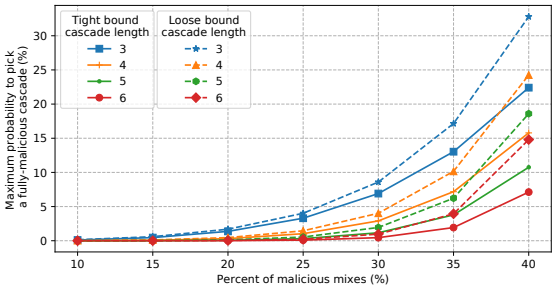


Figure 6: The maximum probability of picking a fully malicious cascade as a function of the cascade length and the power of the adversary.

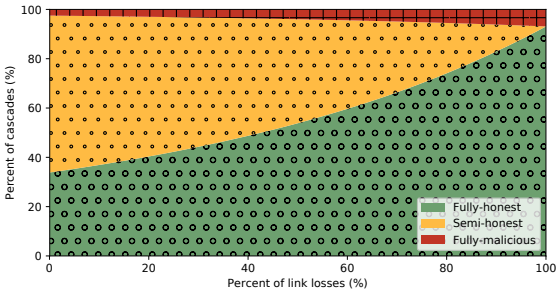


Figure 7: The probability of picking particular classes of cascades after each link loss. The parameters of the simulated mix network are $l = 3$, $n = 100$ and $n_m = 30$.

However, senders *report* these (rare) under-ω receipts to the directories, who would quickly detect that this mix is corrupt. □

7.2 Fully Malicious Cascades Attacks

If the packets are relayed via a *fully malicious cascade*, an adversary can trivially track them. Consequently, adversaries would like to divert as much traffic as possible to the fully malicious cascades. Attackers can try to maximize their chances by: (1) increasing the probability that fully malicious cascades are included in the set $\mathcal{C}$ produced by the directory authorities during the inter-epoch process, and/or (2) increasing the probability that clients pick a fully malicious cascade from $\mathcal{C}$ during an epoch.

Because cascades are chosen uniformly over all valid cascades, the only way the adversary can influence the cascades generation process is by excluding semi-honest cascades. However, they can only exclude cascades by dropping links they are a part of, therefore, the adversary cannot exclude any honest links or honest mixes[6], meaning they cannot exclude any fully honest cascades. However, adversaries are able to disconnect semi-honest cascades by disconnecting semi-honest links and thereby increase the probability of picking a fully malicious cascade. Interestingly, we found that such an attack only slightly increases the chance of selecting a fully malicious cascade – while significantly increasing the chance of selecting a fully honest cascade (see Claim 5). Further, this strategy makes it easier to detect and eliminate sets of connected adversarial domains (see section 6).

Claim 5. *Let C_{Adv} denote a set of fully malicious cascades. The maximum probability to pick a fully malicious cascade during cascades generation process, after the semi-honest cascades were excluded by the adversary is*

$$\Pr(c \in C_{Adv}) \leq \left(\frac{n_m}{n_h - l + 1}\right)^l.$$

Argument. See [34].

Figure 6 and Figure 7 present the probability of picking a fully malicious cascade depending on the number of mixes colluding with the adversary and the percentage of lost links.

Once n_c cascades are generated, the adversary could try to bias the probability of clients choosing a fully malicious cascade. To do so, the adversary can sabotage semi-honest cascades [9] through dropping messages, and in an extreme case, exclude them all. We illustrate in Figure 8 the attack cost, expressed as the number of links the adversary must affect in order to achieve a certain probability of success in shifting clients to a fully malicious cascade. Note, that the larger the number of cascades $\mathsf{n_c}$, the more expensive the attack, and the lower the probability of success.

7.3 Security of Loop Messages

Since loop messages are generated and processed in the same way as genuine messages, the binary pattern does not leak any information. However, adversaries can still seek ways to predict when loop messages are sent; for example, by observing the timing pattern and the rate of sent messages.

Detecting loop messages. Adversaries can try to guess whether a particular message is a loop message or not. A successful guess allows the adversary to drop non-loop messages without being detected, while still sending receipts for them to the previous mix. We formulate the following claim:

Claim 6. *Assume that an adversary that does not control the last mix in the cascade, drops a packet. The probability of this message being a loop message sent by a non-malicious client is at least α.*

Argument. It suffices to consider packets sent by non-malicious clients. When a non-last mix receives such pack-

[6] Even if all adversarial mixes disconnect from an honest mix, it is still not enough for exclusion, since $thresh > n_m$.

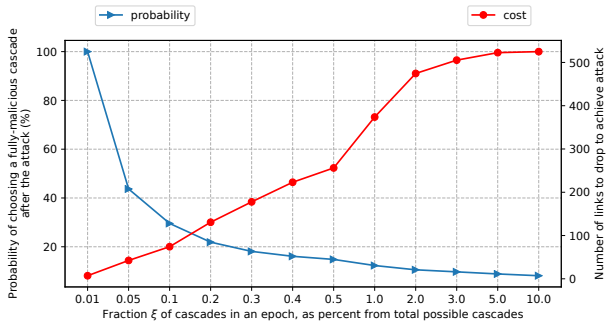


Figure 8: The costs (red, right axis) and success probability (blue, left axis) of performing DoS [9] attacks based on the fraction of cascades active in every epoch. Cost is measured in links the adversary must sacrifice; as Figure 9 shows, even the minimal 'cost' essentially implies detection of *all* active corrupt mixes. Furthermore, using just 1% of the possible cascades suffices to reduce success probability to about 10% or less.

ets, it does not know the destination. Furthermore, as described in section 4.3, loop packets are sent by non-malicious clients according to the rate defined by α of genuine traffic and are bitwise indistinguishable from genuine packets. Hence, even if the mix would know the identity of the sender, e.g., by being the first mix, the packet can still be a loop message with probability at least α. □

Note that a malicious non-last mix that drops a loop message, yet sends a receipt for it and remains connected to the next mix, would be proven malicious and excluded from the network. On the other hand, if such mix does not send a receipt, then it loses a link.

Malicious last mix. Claim 6 does not address the last mix. There are two reasons for that: first, in contrast to mixes, clients do not send receipts back to mixes. Therefore, a last mix cannot prove it actually delivered the packets. Secondly, the last mix may, in fact, identify non-loop messages in some situations. For example, if a client did not send packets in round r, then all the packets it is about to receive in round $\mathsf{r}+x$ (where x is the number of rounds it takes to complete a loop) are genuine traffic sent by other clients. Therefore, these messages can be dropped without detection.

However, dropping of messages by the last mix can also be done against the *ideal mix* (see Section 2.4), e.g., by a man-in-the-middle attacker. In fact, similar correlation attacks can be performed even without dropping packets, if clients have specific sending patterns. Therefore, mitigating this attack is beyond Miranda goals, and should be handled by the applications adopting Miranda [7].

[7] For example, [24, 44] use fixed sending rate (thus, foiling the attack). A concerned client can simply make sure to send additional loop packets in every round where no genuine traffic is relayed.

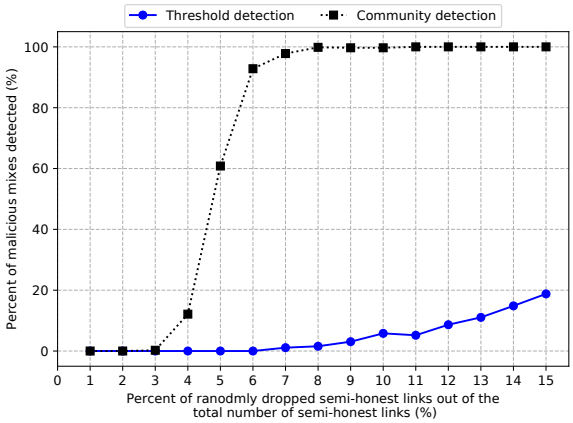


Figure 9: The effect of using community detection against malicious mixes.

7.4 Evaluation of Community Detection

The discussion in Section 6 presented several community detection techniques to leverage Miranda's reported links information into a detection tool that removes malicious mixes from the system. We now argue that the contribution of these mechanisms is both important and secure.

7.4.1 Empirical Results

We implemented the *Threshold Detection* and *Community Detection* algorithms described in Section 6, and evaluated them as follows. We generated a complete graph of $n = 100$ mixes where $n_m = 33$ of them are malicious. We modeled a random adversary by randomly dropping a fraction of the semi-honest links, making sure that any mix does not drop more than or equal to $thresh = n_m + 1$ links.

Figure 9 demonstrates the effectiveness of the algorithms. The *Threshold Detection* algorithm starts to become effective when roughly 10% of the semi-honest links are reported and improves as the number of reports increases. In comparison, the *Community Detection* algorithm presents significantly better results, starting when 4% of the semi-honest links are dropped and after 8% the algorithm is able to expose all possible malicious mixes. Considering that the *Community Detection* algorithm can only operate on malicious mixes that dropped more than one link, these results show that the algorithm effectively mitigates the non-strategic adversary. In [34], we discuss and compare another possible community detection algorithm, which potentially yields even better results.

7.4.2 Security Analysis

In essence, both the *Threshold Detection* algorithm and the *Community Detection* algorithm do the same thing: they both remove malicious mixes from the system. Therefore,

the only way for a strategic adversary to abuse these algorithms is to strategically drop links in a way that causes these algorithms to wrongfully remove honest mixes from the system, due to misclassification of honest mixes as malicious. We now argue that the *ThresholdDetection* and *CommunityDetection* algorithms are secured against such attack.

Claim 7. *An honest mix $M_i \in \overline{\mathcal{G}}$ never satisfies $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) \geq thresh$.*

Proof. Assume to the contrary that there exists an honest mix $M_i \in \overline{\mathcal{G}}$ that satisfies $\mathsf{Deg}(M_i) \geq thresh$. However, if this is the case, then $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) \geq thresh$, which implies $\mathsf{Deg}_{\mathcal{G}}(M_i) \leq n - thresh \leq n_h - 1$, which means that at least one honest mix disconnected from M_i, contradicting the assumption that honest links never fail. □

Claim 8. *The ThresholdDetection algorithm never removes honest mixes.*

Proof. According to the implementation of *Threshold Detection*, the algorithm only removes mix $M_i \in \overline{\mathcal{G}}$ that satisfies $\mathsf{Deg}(M_i) \geq thresh$. However, following Claim 7, this cannot happen for honest mixes. □

Claim 9. *The CommunityDetection algorithm never removes honest mixes.*

Proof. According to the implementation of *Community Detection*, the algorithm only removes mix $M_i \in \overline{\mathcal{G}}$ that satisfies $\mathsf{Deg}(M_i) > n_m - \mu_i$, which according to Claim 1 never happens for honest mixes. □

8 Related Work

In this section, we place our system in the context of existing approaches and compare Miranda with related works. First, we focus on works that present a similar design to Miranda. Next, we discuss how Miranda improves upon previous mix network designs. Finally, we briefly outline other techniques used to support reliable mixing.

Receipts. The idea of using digitally signed receipts to improve the reliability of the mix network was already used in many designs. In Chaum's original mix network design [12] each participant obtains a signed receipt for packets they submit to the entry mix. Each mix signs the output batch as a whole, therefore the absence of a single packet can be detected. The detection that a particular mix failed to correctly process a packet relies on the fact that the neighbouring mixes can compare their signed inputs and outputs. Additionally, [12] uses the untraceable return addresses to provide end-to-end receipts for the sender.

Receipts were also used in reputation-based proposals. In [20], receipts are used to verify a mix failure and rank their reputation in order to identify the reliable mixes and use them for building cascades. The proposed design uses a set of trusted global witnesses to prove the misbehavior of a mix. If a mix fails to provide a receipt for any packet, the previous mix enlists the witnesses, which try to send the packet and obtain a receipt. Witnesses are the key part of the design and have to be engaged in every verification of a failure claim, which leads to a trust and performance bottleneck. In comparison, Miranda does not depend on the witnesses, and a single one is just used to enhance the design. Moreover, in [20] a failure is attributed to a single mix in a cascade, which allows the adversary to easily obtain high reputation and misuse it to de-anonymize clients. Miranda rather than focusing on a single mix, looks at the link between the mixes.

In the extended reputation system proposed in [22] the reputation score is quantified by decrementing the reputation of all nodes in the failed cascade and incrementing of all nodes in the successful one. In order to detect misbehaviors of malicious nodes, the nodes send *test messages* and verify later via a snapshot from the last mix, whether it was successfully delivered. Since the test messages are indistinguishable, dishonest mixes risk being caught if they drop any message. However, the penalty for dropping is very strong – if a single mix drops any message, the whole cascade is failed. Therefore, because a single mix's behavior affects the reputation of all mixes in the cascade, the malicious nodes can intentionally fail a cascade to incriminate honest mixes. This design also proposed the *delivery receipts*, which the recipient returns to the last mix in the cascade in order to prove that the message exited the network correctly. If the last mix is not able to present the receipt, then the sender contacts a random node from the cascade, which then asks the last mix to pass the message and attempts to deliver the message.

Trap messages and detecting active attacks. The idea of using trap messages to test the reliability of the network was discussed in many works. The original DC-network paper [11] suggested using *trap* messages, which include a safety contestable bit, to detect message disruption. In contrast, the flash mixing [28] technique, which was later proved to be broken [38], introduces two dummy messages that are included in the input, and are later de-anonymized after all mixes have committed to their outputs. This allows the participants to verify whether the mix operation was performed correctly and detect tampering. However, both of those types of trap messages are limited to these particular designs.

The RGB-mix [18] mechanism uses heartbeat *loop* messages to detect the (n-1) attacks [45]. Each mix sends heartbeat messages back to itself, and if the (n-1) attack is detected the mix injects cover traffic to confuse the adversary. However, the key assumption of the proposed mechanism is limited only for anonymity among mix peers.

Mixmaster [39] and Mixminion [14] employed an infrastructure of *pingers* [43], special clients sending probe traffic

through the different paths in the mix network and recording publicly the observed reliability of delivery. The users of the network can use the obtained reliability statistics to choose which nodes to use.

Recent proposals for anonymous communication have also employed built-in reliability mechanisms. For example, the new Loopix [44] mix-network system uses *loop cover traffic* to detect (n-1) attacks, both for clients and mixes. However, this idea is limited to detecting only aggressive (n-1) attacks, but mix nodes systematically dropping single packets can operate undetected. Moreover, the authors do not also specify any after-steps or how to penalize misbehaving mixes.

The Atom [33] messaging system is an alternative design to a traditional mix networks and uses *trap* messages to detect misbehaving servers. The sender submits *trap* ciphertext with the ciphertext of a message, and later uses it to check whether the relaying server modified the message. However, the trap message does not detect which mix failed. Moreover, Atom does not describe any technique to exclude malicious servers, and a failed trap only protects against releasing the secret keys.

Other approaches. The literature on secure electronic elections has been preoccupied with reliable mixing to ensure the integrity of election results by using zero-knowledge proofs [3, 6, 29] of correct shuffling to verify that the mixing operation was performed correctly. However, those rely on computationally heavy primitives and require re-encryption mix networks, which significantly increase their performance cost and limits their applicability. On the other hand, the more 'efficient' proofs restrict the size of messages to a single group element that is too small for email or even instant messaging.

An alternative approach for verifying the correctness of the mixing operation were mix-nets with randomized partial checking (RPC) [30]. This cut-and-choose technique detects packet drops in both Chaumian and re-encryption mix-nets, however, it requires interactivity and considerable network bandwidth. Moreover, the mix nodes have to routinely disclose information about their input/output relations in order to provide evidence of correct operation, what was later proven to be flawed [32].

9 Limitations and Future Work

Challenges for making Miranda practical. The Miranda design includes several significant simplifying assumptions, mainly: (1) fixed set of mixes, (2) majority of benign mixes, (3) reliable communication and processing, and (4) synchronized clocks. Such assumptions are very limiting in terms of practical deployment; practical systems, e.g., Tor, cannot 'assume away' such issues. Future work should try to avoid these assumptions, while maintaining tight security analysis and properties as done in Miranda, or identify any inherent trade-offs.

Avoiding the clock synchronization assumption seems easy - simply adopt a secure clock synchronization protocol. However, avoiding the other three assumptions ((1) to (3)) seems much more challenging.

First, consider assumptions (1) and (2), i.e., assuming a *fixed set of mixes with majority of benign mixes*. These assumptions are central to Miranda design; since the goal of Miranda is to provide a way to penalize active attackers. If the adversary can simply retire penalized malicious nodes and replace them with new nodes that have an untarnished reputation, then there is no real gain in even trying to penalize or expose the adversary, and it becomes hard to argue why we can even assume most mixes are benign. However, a practical mixnet must allow a dynamic set of mixes, for both scalability and churn - mixes joining and leaving over time.

Next, consider the third assumption: *reliable communication and processing*. In practice, communication and processing failures will definitely happen - in particular, as a result of intentional DoS attacks. We believe that future work may deal with this significant challenge by both *minimizing failures*, by designing robust underlying mechanisms such as highly-resilient transport layer; and *refined assumptions and analysis*, e.g., considering incentives and game-theory analysis, to ensure that the system is robust to 'reasonable' levels of failures.

These issues are significant challenges for future research, essential towards the implementation of Miranda in practical systems. For example, such research must develop a reasonable model to allow nodes to join (or re-join), without allowing the adversary to gain majority by adding many mixes, as in Sybil attacks, and to retain the impact of removing corrupt mixes.

Extension to Continuous Mixnet. Miranda is designed for a synchronous mixnet. Recent research in mix networks showed that continuous-time mixes, especially pool mixes, may allow anonymity for low latency communication [44]. Future work may investigate how to integrate Miranda with continuous mixnets such as Loopix [44]. Such integration would raise challenges, such as, how would a mix know when it should receive the response from the next mix, esp. without leaking information to an attacker.

10 Conclusion

In this work, we revisited the problem of protecting mix networks against active attacks. The analysis performed showed that active attacks can significantly increase the adversary's chances to correctly de-anonymize users. Miranda achieves much better efficiency than previous designs, but at the same time quickly detects and mitigates active adversaries. Miranda employs previously studied techniques such as packet

receipts and loop traffic alongside novel techniques to ensure that each dropped packet penalizes the adversary. We take a new approach of focusing on problematic links between mixes, instead of mixes themselves. We also investigate how community detection enhances our mechanism effectively. The overall contribution of our work is an efficient and scalable detection and mitigation of active attacks. For additional details, including implementation details and efficiency, see [34].

Acknowledgments

We are grateful to our shepherd, Roger Dingledine, and to the anonymous reviewers, for their helpful and constructive feedback. This work was partially supported by the IRIS Grant Ref: EP/R006865/1 and by an endowment from the Comcast corporation. The opinions expressed in the paper are those of the researchers themselves and not of the universities or sources of support.

References

[1] Nym technologies, 2019. `https://nymtech.net/`.

[2] Panoramix project, 2019. `https://panoramix.me/`.

[3] Masayuki Abe. Mix-networks on permutation networks. In *International Conference on the Theory and Application of Cryptology and Information Security*, 1999.

[4] Dakshi Agrawal and Dogan Kesdogan. Measuring anonymity: The disclosure attack. *IEEE Security & Privacy*, 2003.

[5] Anonymous. QuicR: extending Quic for resiliency to extreme packet losses, 2019. Available from the authors.

[6] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012.

[7] Mihir Bellare, Juan A. Garay, and Tal Rabin. Distributed pseudo-random bit generators : A new way to speed-up shared coin tossing. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1996.

[8] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free mix routes and how to overcome them. In *Designing Privacy Enhancing Technologies*. Springer, 2001.

[9] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[10] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The Guardian*, 2018.

[11] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology, Springer*, 1988.

[12] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1981.

[13] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. 2015.

[14] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*, 2003.

[15] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *30th IEEE Symposium on Security and Privacy (S&P)*, 2009.

[16] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross J. Anderson. Sybil-resistant DHT routing. In *10th European Symposium on Research in Computer Security ESORICS*, 2005.

[17] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*, 2009.

[18] George Danezis and Len Sassaman. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, 2003.

[19] Harry Davies. Ted Cruz using firm that harvested data on millions of unwitting Facebook users. 2015.

[20] Roger Dingledine, Michael J Freedman, David Hopwood, and David Molnar. A reputation system to increase mix-net reliability. In *International Workshop on Information Hiding*, 2001.

[21] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. Synchronous batching: From cascades to free routes. In *International Workshop on Privacy Enhancing Technologies*, 2004.

[22] Roger Dingledine and Paul Syverson. Reliable MIX cascade networks through reputation. In *International Conference on Financial Cryptography*, 2002.

[23] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 1983.

[24] Nethanel Gelernter, Amir Herzberg, and Hemi Leibowitz. Two cents for strong anonymity: the anonymous post-office protocol. 2018.

[25] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *Advances in Cryptology—EUROCRYPT*, 1996.

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987.

[27] Glenn Greenwald and Ewen MacAskill. NSA Prism program taps in to user data of Apple, Google and others. 2013.

[28] Markus Jakobsson. Flash mixing. In *Proceedings of the 18th ACM symposium on Principles of distributed computing*, 1999.

[29] Markus Jakobsson and Ari Juels. Millimix: Mixing in small batches. Technical report, DIMACS Technical report, 1999.

[30] Markus Jakobsson, Ari Juels, and Ronald L Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, 2002.

[31] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 2001.

[32] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In *RSA Conference*. Springer, 2013.

[33] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[34] Hemi Leibowitz, Ania Piotrowska, George Danezis, and Amir Herzberg. No right to remain silent: Isolating malicious mixes - full version. `https://eprint.iacr.org/2017/1000`.

[35] Shengyun Liu, Christian Cachin, Vivien Quéma, and Marko Vukolic. Xft: practical fault tolerance beyond crashes. 2015.

[36] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[37] Ralph Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO*, 1987.

[38] Masashi Mitomo and Kaoru Kurosawa. Attack for flash mix. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.

[39] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster protocol – version 2. *IETF Draft*, 2004.

[40] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

[41] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *IEEE Symposium on Security and Privacy*, 2005.

[42] Lasse Overlier and Paul Syverson. Locating hidden servers. In *IEEE Symposium on Security and Privacy*, 2006.

[43] Peter Palfrader. Echolot: a pinger for anonymous remailers, 2002.

[44] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix anonymity system. In *26th USENIX Security Symposium*, 2017.

[45] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, 2002.

[46] Victor Shoup. Practical threshold signatures. In *International Conference on the Theory and Application of Cryptographic Techniques*, 2000.

[47] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *Designing Privacy Enhancing Technologies*. Springer, 2001.

[48] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP*. ACM, 2015.