Hemi Leibowitz*, Ania M. Piotrowska, George Danezis, and Amir Herzberg

No right to remain silent: Isolating Malicious Mixes

Abstract: Mix networks are a key technology to achieve network anonymity, private messaging, voting and database lookups. However, simple mix networks 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*, a synchronous mix network mechanism, which is *provably secure* against malicious mixes attempting active attacks to de-anonymize users, while retaining the simplicity, efficiency and practicality of mix networks designs. Miranda derives a robust mix reputation through the first-hand experience of mix node unreliability, reported by clients or other mixes. As a result, each active attack – including dropping packets – leads to reduced connectivity for malicious mixes and reduces their ability to attack. We show, through experiments, the effectiveness and practicality of Miranda by demonstrating that attacks are neutralized early, and that performance does not suffer.

DOI foobar

1 Introduction

Mix networks [8] are an established method for providing anonymous communication between senders and recipients. While a single honest mix in a cascade is enough to ensure anonymity against global passive attackers, classical mix networks 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 [2]. Similarly, denial-of-service (DoS) attacks can be used to enhance de-anonymization [6], and $(n-1)$ attacks allow tracing packets over honest mixes [42].

The problem of strengthening decryption mix networks, in order to improve the reliability and detection of active attacks, is a challenging task, since the goal is 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 network. Therefore, several approaches to the problem of active attacks and reliability were studied, however they have many 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 present *Miranda*[1], a practical and efficient reputation-based decryption mix network, which allows to detect and isolate malicious nodes, together with a security argument to demonstrate its effectiveness against active attacks. Our design includes a set of secure and decentralized *mix directory authorities*, for selecting and distributing cascades once every *epoch*. These cascades are selected based on evidence of faulty links between mixes. Miranda employs a novel approach of examining links between mixes, instead of focusing on the mixes themselves. By carefully gathering evidence of misbehaving links, Miranda disconnects them. Consequently, the adversary loses the necessary resources for attacks in long-term. Repeated misbehaviors result in the complete exclusion of the misbehaving mixes from the system, see Figure 1.

Contributions. Our paper makes the following contributions:

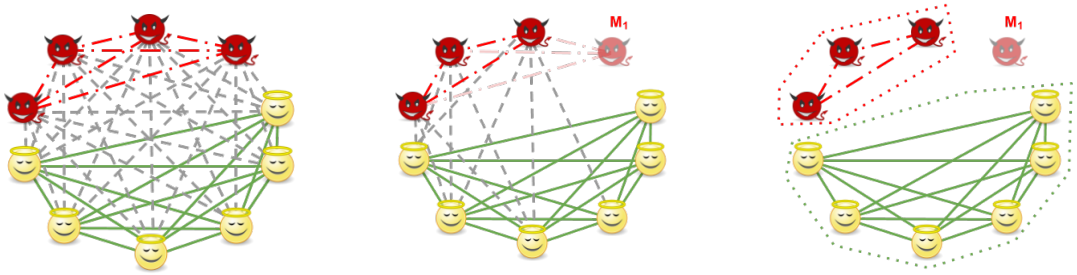
- We present Miranda, an efficient and scalable mechanism that detects and mitigates active attacks. To

***Corresponding Author: Hemi Leibowitz:** Bar-Ilan University, IL

Ania M. Piotrowska, George Danezis: University College London, UK

Amir Herzberg: Bar-Ilan University, IL and University of Connecticut, US

1 "Miranda warning" is the warning used by the US police, in order to notify people about their rights before questioning them. Since Miranda's design and mechanisms prevent adversaries from silently (but actively) attacking the mix network, we refer to it as *no right to remain silent*, hence the name.



(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 3.2.3). A malicious node M_1, which lost more links than a pre-defined threshold, is completely excluded from the system.

(c) Miranda applies community detection (section 5.2) to detect the group of dishonest nodes and disconnect the entire group from the honest nodes.

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

protect against such attacks we leverage reputation and local reports of faults. Miranda is applicable to practical implementation and can be integrated with 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 community detection in a novel way, which further improves its effectiveness.
- We implement and evaluate the performance and scalability properties of Miranda. Our prototype shows high performance for decryption mix networks and the low overhead of the proposed defenses.

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 the design of Miranda. In Section 4 and 5, we detail two main protocols of Miranda, which detect and penalize active attacks. In Section 6, we evaluate the security properties of Miranda against active attacks. In Section 7, we evaluate the performance and resistance to DoS attacks of Miranda. In Section 8, we contrast our design to related work. Finally, we conclude in Section 9.

2 System Model and Motivation

In this section, we outline the general model of the mix network system, define the threat model and summarize the security goals of Miranda. We also motivate our work by quantifying how active attacks threaten anonymity in mix network systems.

2.1 General system model

We consider an anonymous communication system consisting of set $\mathcal{U}$ of users communicating over the *decryption mix network* [8] operating in synchronous batches, denoted as *rounds*. Depending on the path constraints, the topology may be arranged in separate cascades or a Stratified network [15]. As $\mathcal{M}$ we denote the set of all mixes building the anonymous network. Each message is 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 [10]. 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 **Pack**(path, routingInfo, rnd, recipient, message), where rnd 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. Moreover, all malicious entities in the system collude with the adversary, giving access to their internal states and keys. While the adversary might control a significant percentage of the participating entities, we assume that there is still an honest majority of nodes[2]. As n we denote the total number of mixes in the network, n_m of which are malicious and n_h are honest ($|\mathcal{M}| = n$). 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. As *semi-honest* link we call a link between an honest and malicious mix.

The adversary may launch active attacks by dropping or delaying packets, to facilitate traffic analysis attacks. However, the adversary is limited in terms of active network attacks, as 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 [32] that assumes honest nodes can eventually communicate synchronously. It allows more efficient Byzantine fault tolerance schemes, such as the one we present. However, our evaluation does take into consideration some messages between honest mixes being dropped, as a result of realistic network-level attacks.

2.3 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 Appendix A. Our results support the findings of previous works on statistical disclosure attacks [2] and DoS based attacks [6], and 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 only a small number of active attacks. Further, we show how Miranda succeeds in achieving 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, Miranda 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 unreliable links or misbehaving mixes. Miranda offers the following protections against active attacks:

2 In practice, there are probably multiple non-colluding adversaries, each controlling its portion of nodes. While the total number of malicious nodes among all adversaries might exceed the total number of honest nodes, there are still more honest nodes than any single adversary's nodes.

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 Miranda's Design

In this section, we present an overview of the Miranda design, and how it integrates with the mix network model presented in subsection 2.1.

3.1 Directory authorities

Mix network management is secured by introducing *directory authorities*, a set of d semi-trusted servers that maintain a list of available mixes and links between them. We assume that a fraction d_m of authorities can be malicious and collude with the adversary. As d_h we denote the number of honest authorities ($d = d_m + d_h$). Miranda operates in epochs, denoted by E. During each epoch, users communicate over the mix network and report any misbehavior they encounter to the directory authorities. The *directory authorities* are responsible for processing these reports between epochs. During this *inter-epoch* phase, they generate a set of new cascades to be used in the next epoch. The newly generated cascades will reflect all denunciations about mixes' misbehaviors, reported by clients or mixes. Namely, cascades exclude links that were reported, or mixes involved in too many reports. 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. To ensure that malicious mixes alone are unable to exclude honest mixes, we assume that $\mathsf{thresh} > n_m$.

3.2 The Miranda protocol overview

3.2.1 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.

3.2.2 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 [29] statement confirming that a packet p was received by mix M_i. Receipts must be sent and received by the proceeding 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 $\mathsf{receipt} \leftarrow \mathbf{Sign}(\mathsf{p} \mid\mid \mathsf{receivedFlag} = 1)$, where $\mathbf{Sign}(\cdot)$ is a secure digital signature algorithm, and $\mathbf{Verify}(\cdot)$ 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 [34] 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

3 Although **Sign** and **Verify** use the relevant cryptographic keys in their operation, we abuse notations and for simplicity write them without the keys.

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: $\mathsf{receipt} = (\sigma_{\mathsf{C}}, \Gamma_{\mathsf{p}}, \mathsf{r})$, 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 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: $\mathsf{receipt} = (\sigma_i, \mathsf{r})$, 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 it was delivered correctly.

Mixes record the receipts and confirmations to prove later that they behaved honestly in the mixing operation. However, to ensure that messages are not trivially tracked through low volume cascades, mixes only generate receipts if they received enough packets. Namely, if a mix relays to another mix less than ω packets, where ω is the minimum number of packets allowed, the subsequent mix does not generate receipts and does not relay these messages.

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 honest 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 [17], however, to our knowledge this approach was not yet applied to the problem of mix network reliability.

3.2.3 Detecting active attacks

To deter active attacks, clients periodically, yet randomly, craft and send *loop messages* to themselves. In order to construct a *loop message*, client S chooses a unique random bit-string K_S and a random cascade C. Loop messages are encoded in the same manner as regular messages, making them *indistinguishable* from other messages at any stage of their routing. The *loop message* is encapsulated into the packet format as follows

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

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, no one may 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. 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 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.

The rate of loop messages is adjusted according to α, which is the expected probability of detection desired by Alice. 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 $\lceil \beta \cdot \alpha \rceil$ additional loop messages are sent alongside the genuine messages. However, this only ensures α in the context of the messages that Alice *sends* and does not ensure the same for the messages she *receives*. Therefore, if γ is the bound on the number of messages that Alice expects in round r and x is the number of rounds that it takes for a loop message to complete the loop, then Alice sends $\lceil \gamma \cdot \alpha \rceil$ loop messages in round $\mathsf{r} - x$. Since loop messages sent by Alice in round r *also* protect messages received in round $\mathsf{r} + x$, the number of loop messages sent in round r is: $\lceil \mathsf{max}(\beta, \gamma) \cdot \alpha \rceil$. Additionally, the loop

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

messages are injected into the network following Poisson process, therefore the rate of sending cannot be distinguished from genuine traffic. For analysis of the security of loop messages see section 6.3.

4 Intra-Epoch Process

In this section, we present the mechanisms that operate during an epoch to deter active attacks. We start by describing how active attacks are detected and how this deters malicious behavior. Next, we discuss nodes who refuse to cooperate and conclude with how Miranda deals with sporadic or intentional loss of packets.

4.1 Isolating malicious mixes

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 [33]. 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.

4.1.1 Detecting problematic links or mixes

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 that by returning an appropriate signed response to the client. In case of a loop message that 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. 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 that the resulting packet was indeed forwarded to the subsequent mix or that it disconnected from the "faulty" subsequent mix.

Blaming the neighbors. Malicious mixes performing active dropping attacks would prefer to avoid complete exclusion. One option is to drop a packet without generating a receipt, in order to accuse the previous mix, that it did not forward the packet. This approach causes the preceding mix to immediately disconnect from the malicious one at the end of the round. Alternatively, if the malicious mix decides to drop the packet after it generated an appropriate receipt, the malicious mix must disconnect from the subsequent mix by the end of the round to avoid exclusion. Therefore, an adversary that drops a packet either loses a link or is completely excluded from the network.

Delaying packets. A malicious mix can also delay a packet instead of dropping it, so that the subsequent mix will drop that packet. However, the honest 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 re-

ceipt, the honest mix disconnects from the malicious one. Therefore, the delaying attack costs the mix to either lose a link or to be expelled from the system.

The combination of packet receipts, link disconnection notices, and the isolation process amplify the effect of loop messages. It forces malicious mixes to immediately lose links when they perform active attacks, by either not responding to the preceding mix or recording a disconnection notice about the subsequent mix. Failure to do so in a timely manner, creates potentially incriminating evidence, that would lead to their complete exclusion 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 the full node being excluded from the system with some probability.

4.1.2 Refusing to cooperate

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 directory 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.

Malicious entry mix. If a first mix does not cooperate by refusing to produce a receipt, the client can simply choose 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, clients can force the first mix to cooperate with the help of 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 the witness (and not from the client), which allows isolation process to take place. If the witness itself is malicious, it will also refuse to produce a receipt (otherwise, it loses a link). In that case, the client can simply choose another witness. 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 attack imprudent.

4.2 Handling packet loss

Under real network conditions, packets traversing a network might get dropped. This could be the result of network congestion, mistakes, or due to malicious attacks. Miranda is sensitive to packet losses, because any lost packet is interpreted as a semi-honest link, and results in the exclusion of the link from the system. Hence, an off-path adversary capable of dropping packets in a fully-honest link (e.g., using bandwidth-DDoS [18]), could abuse the mechanism and exclude links between honest mixes. Therefore, it is crucial for Miranda to have a built-in support against sporadic or malicious losses.

One approach to address this challenge is to use TCP, and let its built-in mechanisms to take care of lost packets. This is a good approach if the rate of lost packets is small or if the throughput is not high. However, when TCP detects congestion or packet loss, it drastically reduces its transmission rate. Hence, reduced rate is likely to result in delayed packets, which arrive at incorrect rounds. This causes Miranda to consider the link as semi-honest and exclude it, which is the same as if the packets were intentionally delayed by a malicious mix. For that reason, recovering from losses requires a customized protocol, which, for simplicity of deployment, we implement over UDP. To ensure resiliency to lost packets (either due to benign reasons such as congestion or due to attacks such as DoS attacks), we use a forward error correction (FEC) mechanism [41]. In FEC, packets are batched, and each batch is reinforced with

extra redundant packets. The advantage of FEC is that *any* lost packet can be recovered immediately from the redundant packets, without the need to wait for retransmission. In addition, the redundancy ratio can be adjusted dynamically, to send more/less redundant packets based on the currently estimated loss rate. To deal with (rare) losses in spite of the FEC mechanism, we use retransmission. Namely, in case too many packets are lost and there are not enough redundant packets to allow recovery, the receiver requests the sender to retransmit the missing batch. We emphasize that FEC does not prevent users from overwhelming mixes with more traffic than they can handle. That said, the first mix can mitigate this by limiting the traffic volume from clients, according to the throughput and bandwidth limitations of their next hop. We evaluate how TCP and UDP with FEC and retransmission handle packet loss in section 7.

5 Inter-Epoch Process

In this section, we discuss the inter-epoch operations, taking place between the end of one epoch and the next one. The main goal of this process is to select a new random set of cascades to be used in the coming epoch, avoiding faulty or corrupt links and mixes. For simplicity of presentation, we assume the mix network is not used during the inter-epoch process – although part of the inter-epoch processes may take place concurrently with mixing. The inter-epoch process consists of the following steps.

Propagating disconnections. Directory authorities share amongst themselves the evidence they received, and use it to agree on the set of faulty links and mixes. Evidence consists 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 collected since the previous epoch together with the collective signature of d_m+1 directory authorities. Since evidence was signed by at least one non-faulty directory authority all honest directory authorities eventually have exactly the same set of faulty links, assuming maximum network delay.

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 $\mathsf{thresh} > n_m$), and hence the directories exclude that mix completely. 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 subsection 5.2, we discuss a more advanced filtering technique based on community detection.

While it is sufficient to say that $\mathsf{thresh} = n_m + 1$, such threshold implicitly assumes that honest mixes never fail. A more realistic approach would be to define $\mathsf{thresh} = n_m + y + 1$, where y is the maximum number of honest mistakes or misbehaviors due to massive attacks (e.g., DDoS) that we assume an honest mix would do. Obviously, y must not be too big in order to preserve Miranda's guarantees.

Select and publish cascades. 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. In subsection 5.1, we explain the protocol which directory authorities use to select the cascades, and how it ensures that all honest directory authorities agree on the same set of cascades. To allow clients to easily confirm that they use the correct set of cascades, the directory authorities collectively-sign the set that they determined for each epoch, again using a threshold signature scheme [23, 43]. 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).

5.1 Cascades selection protocol

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 mixes $\mathcal{F}_\mathcal{M} \subset \mathcal{M}$ and 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 $\mathcal{C} \subseteq \mathcal{M}^l$, 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 Appendix B, 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., [24]. However, this is a computationally-expensive process, and therefore, we present a much more efficient alternative. Specifically, all directory authorities 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., [4], among the directory authorities[5].

5.2 Applying community detection

Community detection techniques can be used to extract more information from reports of faulty links and mixes, and tilt the choice of cascades towards honest mixes earlier. We augment the inter-epoch process, by performing an additional step of filtering nodes and propagating the reports of faults to more links and nodes – through a community detection algorithm. The key insight underpinning our approach is that reports of faulty links can only concern links between the honest and malicious set of nodes, and thus they separate the sub-graph into those two types of mixes.

Community detection has been used in previous works to achieve Sybil detection based on social or introduction graphs [11, 12]. However, both our aims and the graph-theoretic assumption we base our analysis on, are very different from those previous works. First, we consider a fixed set of mixes containing at most $\mathcal{F}_{\mathcal{M}}$ corrupt mixes and assume that the problem of Sybil attacks is solved through other means, such as admission control, or resource constraints. Secondly, we make no assumptions on the mixing times of random walks on natural 'social graphs', which is for the best, since those have proven, through empirical studies, to be fragile [37].

Graph, Markov chain, and short walk definition. We consider the graph $\mathcal{G}$ with vertices $M_i \in \mathcal{M}$ representing mixes in the system. We define an edge $(M_i, M_j) \in \mathcal{E}$ to exist between each pair of vertices if neither of M_i, M_j has been reported as faulty, and neither has the link between them been dropped by either mix. We note that the resulting graph is symmetric and undirected. At first, before any reports of faults have arrived at the directory authorities, it is complete since all edges are present. Over time, and as reports of faulty mixes or links arrive, the graph $\mathcal{G}$ becomes more sparse.

We define a Markov chain on the graph $\mathcal{G}$ as a set of probabilistic transitions for all nodes $M_i \to M_j$, that we borrow from SybilInfer [12]. We define as $\mathsf{Deg}(M)$ the degree of the vertice M, and the probability of transiting from two vertices M_i, M_j is:

$$\Pr[M_j|M_i] = \begin{cases} \min\left\{\frac{1}{\mathsf{Deg}(M_i)}, \frac{1}{\mathsf{Deg}(M_j)}\right\} & \text{if} (M_i, M_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

and call the matrix of all those transition probabilities Π; the remaining probability mass from each node is assigned to a self-loop. This transition matrix ensures that the stationary distribution of the Markov walk is uniform across all nodes in connected components of $\mathcal{G}$, as shown in [12]. However, a short random walk, of $\mathcal{O}(\log N)$ steps, will not converge to the stationary distribution for sparser $\mathcal{G}$ since the walks will tend to remain within regions of high capacitance. Similar to the insight underpinning Sybil defenses, the random walks starting from honest nodes tend to remain within the (fully connected) regions of the graph, and the missing links between honest and malicious mixes act as a barrier to those walks escaping in malicious regions of the graph.

We leverage this insight to bias cascade construction. We define $K = \lceil k \cdot \log N \rceil$ where k is a small system constant. Then we compute the transition probability matrix $\Pi^* = \Pi^K$ of a random walk using transitions Π after a short number of steps K. Using the matrix Π^* we can extract the probability that a walk starting at node M_i ends up in any node M_j which we denote as $\pi_i^*[j]$. All directory authorities may compute those distributions deterministically and use the information to infer further faulty links: for any node M_i, we denote as cutoff the smallest probability within π_i^*. Then for any node M_j such that $\pi_i^*[j] < \mathsf{cutoff}$ the directory authorities remove the link between M_i and M_j thus further pruning the graph used to build cascades.

Community detection provides Miranda an important feature. Since adversaries do not wish to be excluded from the system, they do not perform more than thresh attacks from the same mix, thus they distribute

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.

the attacks among several malicious mixes. However, the more attacks they perform, even if these attacks do not exceed the thresh threshold, they increase the probability of excluding the entire malicious collective from the system. Merely the threat of such action is significant in deterring active attacks.

6 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.

6.1 Resisting active attacks

As discussed in subsection 4.1, 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 to 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 1. *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 2. *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 [42] 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 [5], 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

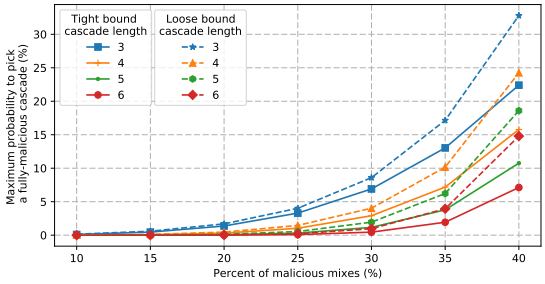


Fig. 2. The maximum probability of picking a fully-malicious cascade as a function of the cascade length and the power of the adversary.

argue that Miranda inflicts serious penalty on the adversary who attempts to perform an aggressive dropping of packets.

Claim 3. *Miranda deters aggressive active attacks.*

Argument. Miranda enforces a minimum number of ω packets required for mixing. Therefore, singling out a specific packet from an entire batch by dropping all packets except the target one is not possible. Even if the adversary generates new messages instead of the dropped packets in order to satisfy ω, the attack can still be detected. In Miranda, a malicious mix that drops packets from another mix, losses a link to a honest mix. A malicious entry mix may drop a packet from the client, but the client then uses a *witness mix* – so the malicious mix either losses a link to the witness mix or the next mix – or has to forward the packet (see subsection 4.1). If the malicious mix drops the packets it acknowledged and instead sends the target message along with enough fake ones, the malicious mix must still sign the Merkle root of the next honest mix in order for the attack to succeed, otherwise the subsequent mix discards the packets and disconnects from the malicious mix. However, this means that clients can prove that the malicious mix received their packet and did not forward it, which results in fully excluding the malicious mix. Note that if a client is the only one that arrives to a cascade with a malicious first mix, the malicious mix can generate enough fake messages for the $(n-1)$ attack to work. However, organically, clients are distributed among all cascades. Therefore, assuming that there are enough clients in the system, this scenario is not likely.

6.2 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 of fully-malicious cascades is 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 4). Further, this strategy makes it easier to detect and eliminate sets of connected adversarial domains (see subsection 5.2).

Claim 4. *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 Appendix C.

Figure 2 and Figure 3 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 $\mathsf{n_c}$ 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 [6] through dropping messages, and in an extreme case, exclude them all. We

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

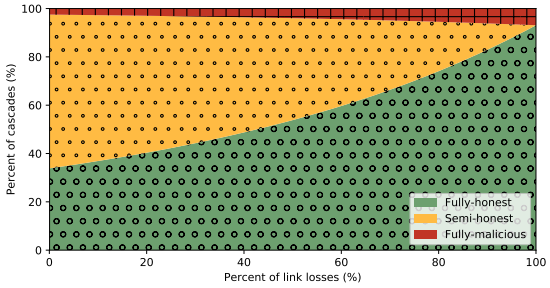


Fig. 3. The probability of picking particular classes of cascades after each link loss. The parameters of the simulated mix network are $l = 3$, $n = 100$ and $m = 30$.

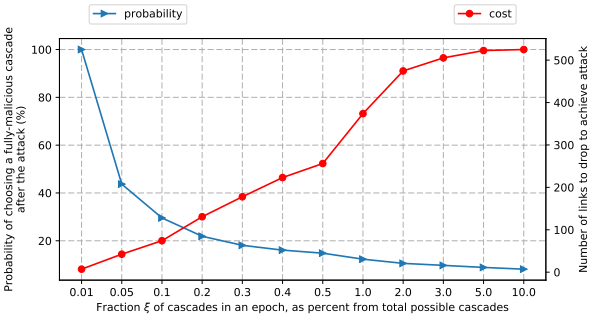


Fig. 4. The cost / success ratio of performing DoS [6] attacks based on the fraction of cascades active in every epoch. Cost (red circle, right axis) is measured in links the adversary must sacrifice; probability of choosing a fully-malicious cascade (blue triangle, left axis) as a result of the attack.

illustrate in Figure 4 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.

6.3 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 5. *Assume that an adversary that does not control the last mix in the cascade, drops a packet. The probability of this message being a non-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 packets, it does not know the destination. Furthermore, as described in subsubsection 3.2.3, 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 identify 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 5 does not address detection of non-loop messages by 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 subsection 2.4). 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 application adopting Miranda [7].

7 For example [22, 40] use fixed sending rate. If the application does not foil the attack, a concerned client can simply make sure that it sends additional loop packets in every round where no genuine traffic is needed to be relayed. Alternatively, while the recipients of the dropped packets are not aware they were dropped, the senders might, if they expect reply message or use the end-to-end ack-based application protocol.

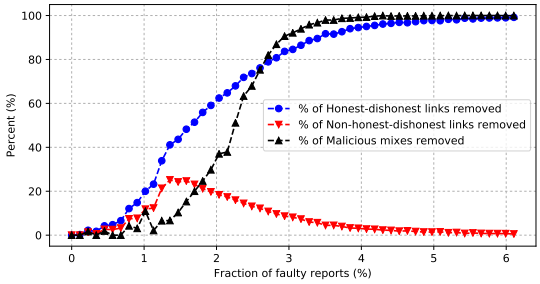


Fig. 5. Effect of the community detection mechanism to detect semi-honest links.

6.4 Evaluation of community detection

We evaluate the community detection based approach through simulations. Given a fraction of reported faulty links, we apply community detection and pruning, and estimate three figures of merit: (1) the fraction of total semi-honest links that are excluded; (2) the fraction of malicious mixes that are detected by pruning those with degree smaller than $n/2$; and (3) the fraction of non-semi-honest links (links connecting two honest nodes, or two malicious ones) that are being removed. The last figure represents the 'false positive' rate of our approach.

We consider a model system with $n = 100$ mixes, out of which 33 are malicious. We perform random walks of length 7, which is the ceiling of the natural logarithm of n. We remove at random a fraction ϕ of distinct reported faulty links, perform community detection, prune links and nodes, and compute the figures of merit above. We consider values for ϕ between 0% and 10% of semi-honest links. The results are illustrated on Figure 5, and each data point is the mean of 20 experiments – error bars are negligible.

We observe that the fraction of semi-honest links ultimately detected by community detection is a large multiple of the faulty links originally reported to the directory authorities: for 1% or originally reported faulty links we can prune about 20% of semi-honest links; for 4% reported we prune over 90% of semi-honest links. Similarly, the number of mixes detected as outright malicious increases very rapidly in the number of reported faulty links, once that information has been enhanced greatly by our community detection: for 2% of reported faulty links we detect over 20% of malicious nodes; for fewer than 4% of reported faulty links we detect over 90% of the malicious nodes. On the other hand, the fraction of honest links mis-categorized and removed first increases with the number of reported faulty links (up to a peak of less than 30% for 1.5% reported links) but then quickly decreases.

Integrated Evaluation. It is worth contextualizing these results in terms of absolute numbers: 6% of reported faulty links – leading to nearly perfect identification of all semi-honest links and malicious nodes – represent merely 270 reports for a network of 100 mixes, out of which 33 are malicious. Achieving the same effect with the simple filtering strategy would require 1122 reports. This is in absolute terms a very small number of loop packets that need to be dropped and isolated until the network can be rid of malicious nodes entirely. Assuming, for example, Miranda requires senders to inject 1% of loop packets to act as a credible detection threat. Taking the scenario we considered in the previous section where each observation of the attacker yields an $\epsilon_\infty = 10^{-2}$, the attacker has a total attack budget of $\epsilon = 2.70$ to expend on attacking clients before all malicious nodes are discovered and eliminated – this is rather small. Even in the case $\lambda = 10$ the total attack budget would be $\epsilon = 27$ across all users.

We conclude that adding community detection to post-process first-hand reports greatly enhances the ability of the system to detect malicious links and leverage those to exclude malicious nodes. However, due to transient misclassification of non-semi-honest links, when the number of reports is low, we recommend that at each inter-epoch processing directory authorities only consider all first-hand reports received – rather than propagating the post-processed information – to avoid compounding errors. Despite being conservative, we show that even after a very small number of first-hand reports we can detect most semi-honest links and malicious nodes.

7 Performance and Resiliency

In this section, we evaluate Miranda's implementation and show that Miranda is indeed capable of processing a large volume of messages; we further show that Miranda is resilient to DoS attacks, which cause high packet-loss rate.

Implementation. We implemented a Miranda prototype in Python and C++, based on Sphinx's Python implementation[8]. All experiments were performed in

8 https://pypi.python.org/pypi/sphinxmix/0.0.7

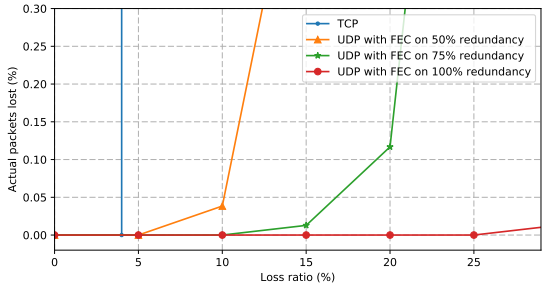


Fig. 6. Packet loss using TCP and UDP with FEC.

the CREATE lab environment[9] (a localized version of Deter-Lab[10]), using Intel Xeon E5-2420 v2 2.20GHz machines with 24 CPUs. Cascades contained 3 mixes which were directly connected, with a round time of 1 minute.

Throughput. We first measure the throughput when there are no packet losses. We compare a Sphinx-only implementation to one with Miranda mechanism, using a Merkle tree of all the messages that arrive during a round. By using the Merkle tree, Miranda can process up to $1.4M$ messages per minute, resulting in both high throughput and reasonable overhead.

Packet loss. To observe how Miranda reacts to packet losses, we compare an implementation that relies on TCP to recover from losses with a UDP version that uses forward error correction (FEC) and retransmission, as discussed in subsection 4.2. TCP is able to recover from losses as long as they are not too substantial. A conservative throughput of $100K$ messages per minute can recover from a loss rate of up to 10%, where a throughput of $1M$ messages per minute can only sustain up to 4% loss. In comparison, in the FEC version the redundancy ratio can be adjusted to support even larger losses, at the cost of more redundant packets being generated and sent. For instance, using a batch size of 16 packets and 100% redundancy (16 redundant packets), Miranda can sustain a loss of up to 25% with a throughput of $1M$ messages per minute (see Figure 6). These results show that Miranda can easily recover from sporadic link losses and can be adjusted to endure even significant losses potentially inflicted by an off-path network adversary.

Still, we obviously cannot guarantee that honest mixes will never be excluded. For example, a completely malfunctioning mix is expected to lose links to honest mixes, resulting in its exclusion. Therefore, it is reasonable to envision that mixes would be able to re-join the system after some time and under some conditions, as long as it will not nullify the exclusion achieved by Miranda.

8 Comparison with Related Work

In this section, we place our system in the context of existing approaches and compare Miranda with the related works. We focus on works that present a similar design to Miranda.

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 [8] 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, [8] uses the untraceable return addresses to provide end-to-end receipts for the sender.

Receipts were also used in the reputation based proposals. The one presented in [14] uses *receipts* 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. Those 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 [14] a failure is attributed to a single mix in a cascade, what 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 [16] 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

9 https://www.create.iucc.ac.il/
10 https://www.isi.deterlab.net/index.php3

penalty for a 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. The idea of using trap messages to test the reliability of the network was discussed in many research works. The original DC-network paper [7] suggested using *trap* messages, which include a safety contestable bit, to detect message disruption. In contrast, the flash mixing [26] technique, which was later proved to be broken [35], 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 [13] mechanism uses heartbeat *loop* messages to detect the (n-1) attacks [42]. 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 [38] and Mixminion [9] employed an infrastructure of *pingers* [39], 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 Loopix [40] mix-network system uses *loop cover traffic* to detects (n-1) attacks, both for clients and mixes. However, this idea is limited to detecting only aggressive (n-1) attacks, but not dropping of single packets. The authors do not also specify how to penalize misbehaving mixes.

The Atom [31] 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 [1, 3, 27] 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) [28]. 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 [30].

9 Conclusion and Future work

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 the 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 a practical, scalable and efficient mechanism for detection and mitigation of active attacks.

We designed Miranda considering its integrity with synchronous mix networks; however, the extension for

other mix network models is also possible. Recent research in mix networks showed that the continuous time mixes are a promising approach towards deployment of mix network systems, which can support low latency communication. Therefore, interesting future work would investigate how to integrate Miranda with continuous time mix models.

References

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

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

[3] 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.

[4] 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 '96)*, New York, USA, 1996.

[5] 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.

[6] 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.

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

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

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

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

[11] 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.

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

[13] 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*.

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

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

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

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

[18] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 2004.

[19] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014.

[20] William Feller. *An introduction to probability theory and its applications*. John Wiley & Sons New York, 1968.

[21] Victor S Frost and Benjamin Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 1994.

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

[23] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In *Advances in Cryptology—EUROCRYPT 96*.

[24] 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 Nineteenth Annual ACM Symposium on Theory of Computing*, 1987.

[25] DP Heyman and MJ Sobel. Superposition of renewal processes. *Stochastic Models in Operations Research: Stochastic Processes and Operating Characteristics*, 2004.

[26] Markus Jakobsson. Flash mixing. In *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*. ACM, 1999.

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

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

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

[30] Shahram Khazaei and Douglas Wikström. Randomized partial checking revisited. In *Cryptographers' Track at the RSA Conference*. Springer, 2013.

[31] 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.

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

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

[34] Ralph Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87*.

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

[36] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[37] Abedelaziz Mohaisen, Huy Tran, Nicholas Hopper, and Yongdae Kim. On the mixing time of directed social graphs and security implications. In *7th ACM Symposium on Information, Computer and Communications Security, ASIACCS*, 2012.

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

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

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

[41] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM*, 1997.

[42] 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.

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

A Information Leakage Analysis

Security game. The challenger chooses a *target cascade* $\mathcal{P}_x$ and gives it to the adversary. The adversary chooses two target senders S_0, S_1 and two target recipients R_0, R_1 who communicate using $\mathcal{P}_x$, and observes the system over multiple rounds. We assume that other clients also communicate using cascade $\mathcal{P}_x$. We call their packets *cover traffic*. In one of the rounds the challenger selects a secret bit b at random. If $b = 0$ then S_0 sends a *challenge message* to R_0 and S_1 sends a *challenge message* to R_1, what we denote as $S_0 \to R_0$ and $S_1 \to R_1$. Otherwise, if $b = 1$, $S_0 \to R_1$ and $S_1 \to R_0$. During the challenge round, the adversary drops a single message of S_0 or S_1 and observes the system. The adversary guesses the value of bit b', and sends b' to the challenger. The adversary wins the game if $b = b'$.

Measurement of adversary's advantage. The adversary observes the volume of traffic injected to the cascade by S_0 and S_1 and the volume of traffic, denoted x_{R_0} and x_{R_1}, received by R_0 and R_1. The adversary examines how the volume of traffic received by R_0 and R_1 is affected by the active attack, and use it to increase the probability of correctly guessing b'. We use a differential privacy metric [19] to bound the likelihood ratio of the observation (x_{R_0}, x_{R_1}) conditioned on the adversary's guess b' using an $\epsilon \geq 0$ and a $0 \leq \delta \leq 1$. Although applying the DP measurement in the context of anonymous channels deviates from its traditional meaning, it is a good measure when we want to investigate the indistinguishability bound on two events observed by the adversary. Intuitively, ϵ defines the maximal leakage the adversary can learn from observing both the events (so how much those events differ), whereas δ is the probability by which the leakage exceeds this ϵ (small ϵ and δ values are better for security). We also consider the volume of the *cover traffic* λ, injected according to the Poisson distribution. We do not use the usual distribution DP does, but a Poisson distribution since sending can be modeled only by a positive distribution. Moreover, Poisson models have been widely used in the computer networks and telecommunications literature [21], since it offers attractive analytical properties. The Poisson distribution is appropriate if the arrivals are from a large number of independent sources, like in networks with many clients and nodes. The superposition of multiple independent Poisson processes results in a Poisson process. Moreover, based on the Palm's Theorem [25] we know that under suitable conditions large number of independent multiplexed streams approach a Poisson process as the number of processes grows. Finally, the memoryless property of Poisson process allows to simplify queuing problems involving Poisson arrivals. However, we highlight that the analysis could be done using other distributions as well.

Let x_{S_0}, x_{S_1} denote the volume of traffic sent by S_0 and S_1 respectively. Similarly, let x_{R_0}, x_{R_1} denote the observed volume of traffic incoming to recipient R_0 and r_1, which can be either from S_0 or S_1 or *cover packets*. Thus, let us define Y_0, Y_1 as random variables, such that $Y_0 \sim Pois(\lambda_0)$ and $Y_1 \sim Pois(\lambda_1)$, which denote the number of cover packets received by R_0 and R_1 respectively (λ_0, λ_1 denote the expected value of the Poisson distribution).

Theorem 1. *Given an observation $O = (x_{R_0}, x_{R_1})$ resulting from a single observation of the adversary performing a dropping attack on a single packet sent by S_b, the relationship of the likelihoods of the observations conditioned on the secret bit b becomes:*

$$Pr[Y_0 = x_{R_0}, Y_1 = x_{R_1} - 1 | b = 0] \leq e^{\epsilon} \Pr[Y_0 = x_{R_0} - 1, Y_1 = x_{R_1} | b = 1] + \delta$$

for $\delta = 1 - \left(\sum_{i=1}^{\infty} CDF_{Y_1}[(1+\epsilon)i] \cdot \frac{\lambda^i e^{-\lambda}}{i!} \right)$,

where CDF denotes the cumulative distribution function of the cover Poisson distribution with rate parameter λ.

Proof. Given the observation $o = (x_C, x_D)$ we consider two cases conditioned by the events that either $b = 0$, i.e., $A \to C$ and $B \to D$, or $b = 1$, i.e., $A \to D, B \to C$. Without the loss of generality we consider a scenario in which the adversary targets sender A. We define a differentially private dependency

$$Pr[Y_C = x_C, Y_D = x_D - 1 | b = 0] \leq e^{\epsilon} \Pr[Y_C = x_C - 1, Y_D = x_D | b = 1] + \delta.$$

Thus, we compute

$$\frac{Pr[Y_C = x_C, Y_D = x_D - 1 | b = 0]}{\Pr[Y_C = x_C - 1, Y_D = x_D | b = 1]} = \frac{\frac{\lambda^{x_C} e^{-\lambda}}{(x_C!)} \frac{\lambda^{x_D - 1} e^{-\lambda}}{((x_D - 1)!)}}{\frac{\lambda^{x_C - 1} e^{-\lambda}}{((x_C - 1)!)} \frac{\lambda^{x_D} e^{-\lambda}}{(x_D!)}} = \frac{x_D}{x_C},$$

Given that, we calculate the values of δ, defined as $\delta = \Pr[Y_D \geq e^{\epsilon} Y_C]$, using the law of total probability and the cumulative distribution function:

$$\begin{aligned} \Pr[Y_D \geq e^{\epsilon} Y_C] &= \sum_{i=1}^{\infty} \Pr[Y_D \geq e^{\epsilon} Y_C | Y_C = i] \Pr[Y_C = i] \\ &= \sum_{i=1}^{\infty} \Pr[Y_D \geq e^{\epsilon} i] \Pr[Y_C = i] \\ &= \sum_{i=1}^{\infty} CDF_{Y_D}[e^{\epsilon} i] \cdot \frac{\lambda^i e^{-\lambda}}{i!}. \end{aligned}$$

□

We also provide a loose (but analytic) bound on δ as a function of ϵ and λ. We compare between the bound (Theorem 2) and the exact calculation (Theorem 1), for different values of λ and a fixed leakage $\epsilon = 0.2$ in Figure 7. As illustrated the below bound is tight for large values of λ, however for small values of λ it does not give us any significant information. Therefore, for small values of λ the formula from Theorem 1 suits better for computing a good approximation of leakage δ.

Theorem 2. *The value of* δ *from Theorem 1 for sufficiently large values of parameter* λ *can be bound as:*

$$\delta \leq \left(\frac{e^{-\epsilon/2}}{(1-\epsilon/2)^{(1-\epsilon/2)}} \right)^{\lambda} + \left(\frac{e^{\epsilon/2}}{(1+\epsilon/2)^{(1+\epsilon/2)}} \right)^{\lambda} + \left(\frac{e^{\frac{\epsilon}{2} - \frac{\epsilon^2}{2}}}{(1 + \frac{\epsilon}{2} - \frac{\epsilon^2}{2})^{(1 + \frac{\epsilon}{2} - \frac{\epsilon^2}{2})}} \right)^{\lambda}$$

Proof. As before, we start by applying the law of total probability and we note, that for small values of ϵ we can approximate $e^{\epsilon} \approx 1 + \epsilon$. Hence,

$$\begin{aligned} &\Pr[Y_D \geq (1+\epsilon) Y_C] \\ &\quad = \sum_{i=1}^{\infty} \Pr[Y_D \geq (1+\epsilon) Y_C | Y_C = i] \Pr[Y_C = i] \\ &\quad = \sum_{i=1}^{\infty} \Pr[Y_D \geq (1+\epsilon) i] \Pr[Y_C = i]. \end{aligned}$$

Thus, we can split the infinite sum into three separate cases as follows

$$\begin{aligned} \Pr[Y_D \geq (1+\epsilon) Y_C] \leq & \underbrace{\sum_{i=0}^{(1-\frac{\epsilon}{2})\lambda} \Pr[Y_C = i] \Pr[Y_D \geq (1+\epsilon) i]}_{(I)} \\ & + \underbrace{\sum_{i=(1+\frac{\epsilon}{2})\lambda}^{\infty} \Pr[Y_C = i] \Pr[Y_D \geq (1+\epsilon) i]}_{(II)} \\ & + \underbrace{\sum_{i=(1-\frac{\epsilon}{2})\lambda}^{(1+\frac{\epsilon}{2})\lambda} \Pr[Y_C = i] \Pr[Y_D \geq (1+\epsilon) i]}_{(III)}. \end{aligned}$$

Note, that for large values of λ the tails of Poisson distribution in parts (I) and (II) are 'heavy', i.e., accumulate a large probability mass. Thus, we can bound those tails by 1 without overestimation. Hence, we obtain

$$\begin{aligned} \Pr[Y_D \geq (1+\epsilon) Y_C] = & \sum_{i=0}^{(1-\frac{\epsilon}{2})\lambda} \Pr[Y_C = i] + \sum_{i=(1+\frac{\epsilon}{2})\lambda}^{\infty} \Pr[Y_C = i] \\ & + \sum_{i=(1-\frac{\epsilon}{2})\lambda}^{(1+\frac{\epsilon}{2})\lambda} \Pr[Y_C = i] \Pr[Y_D \geq (1+\epsilon) i]. \end{aligned}$$

We note that $\Pr[Y_D \geq (1+\epsilon) i]$ in the sum over $i = \{(1 - \frac{\epsilon}{2})\lambda, \ldots, (1 + \frac{\epsilon}{2})\lambda\}$ can be bounded as

$$\Pr[Y_D \geq (1+\epsilon) i] \leq \Pr[Y_D \geq (1+\epsilon)\left(1 - \frac{\epsilon}{2}\right)\lambda].$$

Following this, we have

$$\begin{aligned} &\Pr[Y_D \geq (1+\epsilon) Y_C] \\ &\quad = \Pr[Y_C \leq \left(1 - \frac{\epsilon}{2}\right)\lambda] + \Pr[Y_C \geq \left(1 + \frac{\epsilon}{2}\right)\lambda] \\ &\quad + \Pr[Y_D \geq (1+\epsilon)\left(1 - \frac{\epsilon}{2}\right)\lambda] \sum_{i=(1-\frac{\epsilon}{2})\lambda}^{(1+\frac{\epsilon}{2})\lambda} \Pr[Y_C = i] \end{aligned}$$

Since Y_C is a Poisson distributed variable, and we sum up the probabilities of independent events we can bound the whole sum by 1. Hence,

$$\begin{aligned} &Pr[Y_D \geq (1+\epsilon) Y_C] \\ &\quad \leq \Pr[Y_C \leq \left(1 - \frac{\epsilon}{2}\right)\lambda] + \Pr[Y_C \geq \left(1 + \frac{\epsilon}{2}\right)\lambda] \\ &\quad + \Pr[Y_D \geq (1+\epsilon)\left(1 - \frac{\epsilon}{2}\right)\lambda] \end{aligned} \tag{1}$$

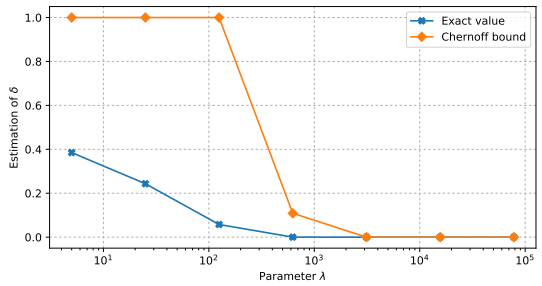


Fig. 7. The precision of upper bound for δ presented in theorem 1 for a fixed $\epsilon = 0.2$. The *exact values* are computed using the importance sampling technique.

Now by applying the Chernoff inequality [36] we can derive a final form of our upper bound for δ[11]:

$$\delta \leq \left(\frac{e^{-\epsilon/2}}{(1-\epsilon/2)^{(1-\epsilon/2)}}\right)^{\lambda} + \left(\frac{e^{\epsilon/2}}{(1+\epsilon/2)^{(1+\epsilon/2)}}\right)^{\lambda} + \left(\frac{e^{\frac{\epsilon}{2}-\frac{\epsilon^2}{2}}}{\left(1+\frac{\epsilon}{2}-\frac{\epsilon^2}{2}\right)^{\left(1+\frac{\epsilon}{2}-\frac{\epsilon^2}{2}\right)}}\right)^{\lambda}.$$

□

Leakage (ϵ,δ) for multiple rounds. The advantage of the (ϵ, δ) leakage quantification presented in Theorem 1 is that it composes under R multiple rounds of dropping and observations. Given the set of observations $\overline{O} = (o_1, o_2, \ldots, o_n)$, where each single observation is defined as $o_i = (x_{C_i}, x_{D_i})$., we compute

$$\frac{\Pr[(x_{C_1}, x_{D_1}), \ldots, (x_{C_R}, x_{D_R})|b=0]}{\Pr[(x_{C_1}, x_{D_1}), \ldots, (x_{C_R}, x_{D_R})|b=1]} = \prod_{i=1}^{R} \frac{\Pr[(x_{C_i}, x_{D_i})|b=0]}{\Pr[(x_{C_i}, x_{D_i})|b=1]} = \prod_{i=1}^{R} \frac{x_{D_i}}{x_{C_i}}$$

From the composition theorem of differential privacy we know that given the value of ϵ, δ for a single round, the likelihood ratio of multiple observations will follow a similar $(\overline{\epsilon_R}, \overline{\delta_R})$ relation, with $\overline{\epsilon_R} = R \cdot \epsilon$ and $\overline{\delta_R} = R \cdot \delta$. However, this estimation of leakage for multiple observations can also be shown to be tragically loose and pessimistic – since it assumes that the worst case occurs in every round. In reality the adversary cannot attain such significant advantage except with negligible probability. Therefore, we focus on analyzing the average case, for which we simulate several observations (x_{C_i}, x_{D_i}) and compute the estimator of the average leakage as

$$e^{R\epsilon} = \prod_{i=1}^{R} \frac{x_{D_i}}{x_{C_i}} \implies \log\left(e^{R\epsilon}\right) = \log\left(\prod_{i=1}^{R} \frac{x_{D_i}}{x_{C_i}}\right)$$
$$\implies \epsilon = \frac{1}{R} \sum_{i=1}^{R} \log\left(\frac{x_{D_i}}{x_{C_i}}\right).$$

This allows us to derive the average case value of the leakage which the adversary can gain after multiple concrete observations (x_{C_i}, x_{D_i}). From the law of large numbers [20] we know, that as R grows, the obtained estimator tends closer to its expected value. And thus:

$$\epsilon_\infty = \lim_{R\to\infty} \hat{\epsilon} = \mathbb{E}[\log Y/X] \text{ for } X, Y \sim \text{Poisson}^{+}(\lambda) \quad (2)$$

where Poisson^{+} denotes the Poisson distribution truncated to only its strictly positive range. The quantity ϵ_∞ represents the expected per-round leakage and thus after R observations we expect the total leakage to be $\epsilon = R \cdot \epsilon_\infty$ However, we note, that if x_C or x_D is 0 the adversary can successfully distinguish who was communicating with whom immediately – representing a catastrophic event for which we cannot bound the leakage under any ϵ. We therefore need to compute the probability of such an event after R observations and fold it within the probability δ. The probability that a Poisson distribution yields zero is $\delta_0 = \Pr[x = 0] = e^{-\lambda}$. Thus after R observed rounds the probability that any such event has occurred is:

$$\delta = 1 - (1 - \delta_0)^{2R} < 2R \cdot \delta_0 \quad (3)$$

Equations (1) and (2) conclude our direct estimation of the (ϵ, δ) for multiple observations. These represent a different trade-off between the two parameters than in the single round analysis: the new δ only represents the catastrophic probabilities any observation is zero – and not the cases where epsilon may be too large as in the single round case.

Evaluating multi-round leakage. Figure 8 shows the values of the leakage estimator ϵ_∞ (estimated using Monte Carlo integration using $10,000$ samples), versus the values of λ. We note that, as the rate of cover traffic λ grows, the leakage significantly decreases. For example, for cover traffic rates of $\lambda = 100$, the rate of leakage $\epsilon_\infty = 10^{-2}$, and thus after $R = 100$ observations we expect a total leakage of $\epsilon = 1$ (following Eq. (1)). Meanwhile $\delta_0 = e^{-100}$ and overall $\delta < e^{-94}$ (from Eq. (2)) which is tiny.

The fact that as the volume of cover traffic increases, the probability δ of a catastrophic event becomes extremely small is comforting. On the other hand, we note that the value of ϵ does grow linearly, and there is a direct inverse relationship (see Figure 8) between the rate of cover traffic each user receives and the rate of round

11 (Note, that the above bound can be made even a little bit tighter, by doing two more precise steps in Equation 1.)

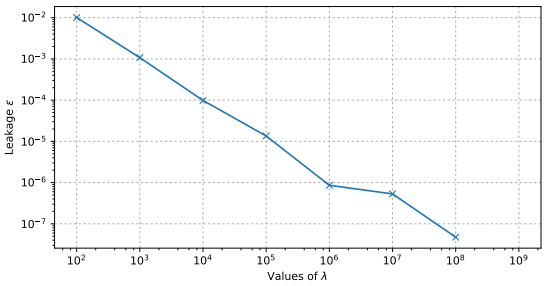


Fig. 8. The comparison of amounts of leakage ϵ_∞ for different values of λ.

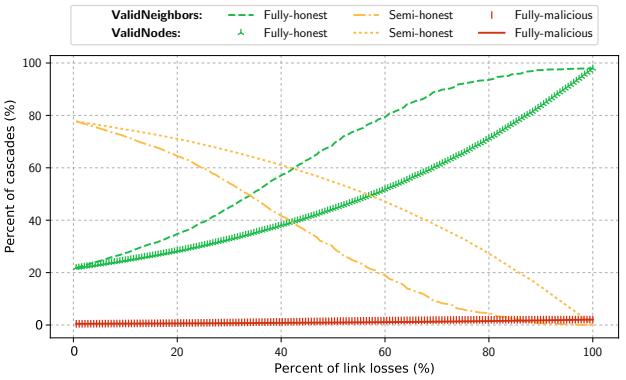


Fig. 9. Probability of picking cascades as function of link losses in ValidNeighbor in comparison to ValidNodes, where $l = 4$ and the adversary controls 30% of the mixes.

leakage. The value of ϵ that can be tolerated in reality depends on the prior belief of the adversary: in the simple cryptographic game proposed the adversary assigns a 50:50 prior likelihood to $b = 0$ or $b = 1$. In a real de-anonymization setting, that prior belief may instead be much lower: for example if the adversary tries to guess which of 100 potential recipients a target sends a message to, the prior belief is as low as 1/100.

B Legitimate-Cascade Predicates

The *cascade selection protocol* described in subsection 5.1, uses several constraints, defined by the *legitimate cascade predicates*, to generate a set $\mathcal{C}$ of valid cascades. Below, we propose a set of such predicates, which validate whether a cascade is legitimate and can be included in $\mathcal{C}$. Depending on the considered predicate, it can either be co-applied jointly with other ones, to eliminate more undesired cascades, or individually.

- $\mathsf{UniqueInCascade}(c) = \{\forall M_i, M_j \in c : \ i \neq j\}$
  Each mix is used only once in a particular cascade c.
- $\mathsf{NonFaulty}(c) = \{\forall M_i \in c : \ M_i \notin \mathcal{F}_\mathcal{M}\}$
  Each mix in cascade c is selected only from the set of non-faulty mixes.
- $\mathsf{OnlyInOneCascade}(c) = \{\forall M_i \in c \wedge \forall\ c' \in \mathcal{C} : \ M_i \notin c'\}$
  Any two cascades should not have a common mix.
- $\mathsf{ValidNeighbor}(c) = \{\forall M_i, M_{i+1} \in c : (M_i, M_{i+1}) \notin \mathcal{F}_\mathcal{L}\}$
  For each pair of directly connected mixes in cascade c, this pair should not be listed in the set of faulty links $\mathcal{F}_\mathcal{L}$.
- $\mathsf{ValidNodes}(c) = \{\forall M_i, M_j \in c : (M_i, M_j) \notin \mathcal{F}_\mathcal{L}\}$
  No two mixes in cascade c can have a faulty link between them.

Other predicates can be defined, however it is important to balance their effect on the system, both in terms of performance and security. Predicates also affect the *penalization factor*, i.e., the price that an adversary pays for losing a link. Consider predicates ValidNeighbor and ValidNodes, where a single link loss excludes a different number of cascades in each approach. In ValidNeighbor, all cascades that contained a dropped link are no longer valid, while in ValidNodes, on top of those cascades, any other cascade that has any two mixes who disconnected from one another is no longer valid. The rationale is that if two mixes are unwilling to directly communicate, they are unwilling to communicate indirectly as well. Therefore, the price that an adversary pays for losing a link significantly increases, as presented in Figure 9, yet increases the chances of choosing a fully-malicious cascade, as presented in Figure 3.

C Fully-Malicious Cascades

In this section, we argue the correctness of claim 4.

Argument. Initially, the probability that a randomly selected cascade is fully-adversarial is $\Pr(c \in C_{Adv}) = \frac{n_m!(n-l)!}{n!(n_m-l)!}$. After the adversary disconnects all semi-honest cascades, the total number of all possible permutations of cascades is $\frac{n_m!}{(n_m-l)!} + \frac{n_h!}{(n_h-l)!}$. Since each cascade is selected uniformly at random the probability of picking a fully-adversarial cascade is defined as

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

□