No right to remain silent: Isolating Malicious Mixes

Hemi Leibowitz
Bar-Ilan University, IL

Ania Piotrowska
University College London, UK

George Danezis
University College London, UK

Amir Herzberg
Bar-Ilan University, IL

ABSTRACT

Mix networks are a key technology to provide network anonymity, used for messaging, voting and private lookups. However, simple mix networks are insecure against malicious mixes, which can drop or delay packets to facilitate traffic analysis attacks. Mix networks with provable robustness address this by using complex and expensive proofs of correct shuffling, which come with a cost and make assumptions that are unnatural to many settings in which mix networks are deployed. We present Miranda, a synchronous mix network mechanism, which is provably secure against malicious mixes – yet retaining the simplicity, efficiency and practicality of mix network designs. Miranda uses first-hand experience of unreliability by mixes and clients, to derive a mix 'reputation', and to ensure that each active attack – including dropping of packets – results in reduction in the connectivity of the malicious mixes, thus reducing their ability to attack. Besides the practical importance of Miranda itself, our results are applicable to other mix networks designs and anonymous communication, and even unrelated settings in which reputation could provide effective defense against malicious participants.

KEYWORDS

Anonymity, mix networks, byzantine attacks

1 INTRODUCTION

Mix networks [5] are an established method for providing communications anonymity for senders and receivers. Typically, a mix network is used by sending messages over a cascade of multiple mixes. A single non-corrupted mix in the cascade suffices, to ensure anonymity against a passive attacker. In the 'classical' mix network design, messages are *layer encrypted*, and each mix removes one layer of the encryption, until the last mix can see the final destination together with the (encrypted) message. Layered-encryption mix networks are simple and efficient, and have been deployed in practical mix network systems, e.g., Mixminion [6] and Mixmaster [30].

However, such 'classical' mix networks are insecure against *active* traffic analysis attacks, often involving *dropping* or *delaying* of packets by malicious mixes. Previous work demonstrates that denial-of-service attacks can be used to enhance de-anonymization in mix networks [4], so-called $n-1$ attacks can be used to trace packets over honest mixes [33], and the pre-conditions for disclosure attacks can be re-created through dropping packets [1].

There have been multiple efforts to extend mix networks, in order to ensure anonymity against malicious mixes. Several proposals use more advanced cryptographic mechanisms, such as zero-knowledge proof of correct mixing [2]. However, those are complex, restrictive and have considerable, possibly prohibitive, overhead. They require re-encryption mix networks to support 'efficient' proofs, which restrict the size of messages to single group elements too small for email, or even media rich instant messaging.

There have been also multiple efforts to secure layered-encryption mix networks against active malicious mixes – and we provide an overview of those efforts as part of our discussion of related work. However, although the impact of active attacks is severe, it is not easy to detect such attacks and even harder to identify and *penalize* the rogue mixes. This is particularly true since malicious mixes may also make false reports of honest mixes being unreliable to exclude them from the network, or to gain any other advantage. Indeed, this direction has been mostly abandoned after several attempts were found vulnerable or to offer insufficient guarantees.

In this work, we return to this basic problem of layered-encryption based mix networks robust to malicious, active mixes. We present *Miranda*, a practical, efficient reputation-based layered-routing mix network, together with analysis showing that it is secure against active, malicious mixes. Our design includes a secure, decentralized *mix directory authorities*, for selecting and distributing cascades once every *epoch*, based on reputation and evidences for faults in specific mixes and links between mixes.

We found that some of the most challenging attacks by rogue mixes involves strategically dropping packets, typically from a specific sender, to detect a recipient which receives less messages – a variant of the disclosure attack. It is a challenge to detect such behavior and identify the attacking mix, without breaking the anonymity. Often, mixes have to drop packets due to congestion control or when decryption fails. Also, the verification of the mixes' behavior often results in contradicting statements.

Miranda deals with such challenges, by carefully providing evidences of misbehavior of a mix. In particular, a mix provides a signed receipt upon receiving a packet, and mixes disconnect from a peer which does not send a receipt. Miranda also detects mixes which fail to forward messages, and they are removed from the mixnet. We refer to these mechanisms as 'no right to remain silent', hence the name Miranda[1].

Contributions This paper makes the following contributions:

Definition of loop packets in decryption mix networks. We propose an encoding for security 'loop packets', that may be used to securely test the network for dropping attacks.

The Miranda mechanism. Miranda is a mechanism that detects and mitigates dropping attacks while retaining the simplicity, efficiency, and scalability of the classical layered-based

[1]The "Miranda warning" is the warning used by the US police, in order to notify people about their rights before questioning them.

mix networks. Hence, it is appropriate for practical implementation and use.

Mix directory authorities. Miranda design explicitly includes the mix network management by a set of directory authorities, who periodically every epoch, select and distribute new cascades.

Use of reputation and evidences against active attacks. Our techniques leverage local reports of faults to defeat active attackers (mixes), may be applicable to other scenarios and problems.

Quantification of attacks. We provide an analysis of dropping attacks in decryption mix networks, and for the first time a security game and qualitative and composable measure of security against dropping attacks.

Effective use of community detection. We show how Miranda can take advantage of community detection in a novel way, which allows significant further improvements in its effectiveness.

1.1 Related Work

Reliability against active, or dropping attacks, in mix networks, has been the subject of theoretical study and has also been addressed in various practical ways. The original mix network design [5] by David Chaum includes facilities for end-to-end receipts, that were however fragile since they potentially leak information about the destination of messages. Deployed mix networks, such as Mixmaster [30] or Mixminion [6], employ an infrastructure of 'pingers' [31], special clients sending probe traffic through the different paths in the mix network, and recording publicly the observed reliability of delivery. The deployed onion router, Tor [14], also operates a measurement infrastructure to establish the reliability and bandwidth of relays [32] by looping or relaying test traffic across the network. Such systems need to be built with care: test messages should be indistinguishable from real traffic, and even the identities of the separate pingers or bandwidth authorities should not betray their function. Otherwise, byzantine mixes may treat test messages differently from normal client messages. Furthermore, there is a challenge of attributing a failure to a specific mix in the cascade or path.

A more formal design approach considers using reputation to hold mixes accountable for message delivery [13] or to engineer reliable cascades over time [16]. There are inherent difficulties in establishing a global trust metric from local and subjective measurements, and our work tries to address this challenge. The literature on mix network attacks has established that active 'trickle and flood' attacks can use packet delaying or blocking to de-anonymize communications [33], and that generic denial-of-service attacks may be used to bootstrap attacks on anonymity by forcing re-transmissions over less secure paths [4]. Active measurements and countermeasures were proposed, e.g., RGB-mix [11]; however, no design was shown to be secure. Furthermore, designs are often assuming unrealistic models, e.g., RGB-mix is designed only for anonymity among peers, i.e., all parties are mixes (no clients) – and this assumption is key to the proposed mechanism.

In contrast, the literature on secure electronic elections has been preoccupied with reliable mixing to ensure the integrity of election results. Early designs such as Flash mixing [21], make use of challenge messages to establish the correctness of mixing, but some were found to be flawed [12]. However, two patterns to ensure reliability have established themselves since the mid-2000: on one hand re-encryption mix cascades may use zero-knowledge proofs (ZKP) to show that their outputs are a secret shuffle of their inputs [2]; on the other hand, Randomized Partial Checking [22, 24] (RPC), a cut-and-choose technique, may be used to detect multiple packet drops with high probability. These techniques have downsides: they have only been fully explored in the context of re-encryption mix cascades; they are computationally demanding in the case of ZKP; and require interactivity and considerable network bandwidth in the case of RPC. These drawbacks seem to make these mechanisms inappropriate for anonymization of messages. For example, Vuvuzela[35], a recent proposal for anonymous messaging, uses 'classical' layered encryption, but at high costs: a single, fixed cascade (non-robust and non-scalable), and use-once dead-drops, resulting in high overhead (must check every possible dead-drop).

Miranda leverages local trust decisions, derived from direct experience, and uses that experience to exclude paths containing malicious mixes. The use of graphs of trust decisions to establish trusted and untrusted regions has been proposed before, in the context of Sybil defenses [9, 36], and based on fast mixing assumptions. However, empirical works have questioned the validity of the fast mixing assumption in social networks [29]. In this work, we do not rely on fast mixing properties of social graphs.

2 MODEL AND PRELIMINARIES

2.1 System Model

The Miranda system enhances anonymity in decryption mix networks [5], against malicious mix relays, by detecting adversarial delaying or dropping of mix packets. The key entities in the mix network are *clients*, *mixes* and a set of *directory authorities*, which we define as follows:

Client: A software application used to send messages between users, using the anonymous communications infrastructure. We entrust honest clients to also send special *loop messages* to themselves through the mix network, that they can leverage in order to identify mix misbehavior.

Mixes: A set of synchronous timed mixes, arranged in cascades of fixed length l, that decode and shuffle all packets that arrive at round T_i, and forward them to their next-hop during round T_{i+1}. Honest mixes provide signed receipts for all the packets they receive, and retain receipts for packets forwarded from the next mix in their cascade; and report mixes that are unresponsive.

Directory Authorities: A set of semi-trusted servers, that maintain a list of available mixes and links between them, and, once every *epoch*, collaboratively generate cascades of l mixes, based on available links, for the use of clients.

2.2 Threat Model

We assume that all malicious mixes and authorities collude with an adversary that aims to either deanonymize one or more messages traveling through the mix network, or aims to disrupt service

through denial-of-service attacks. We note that there is a well-established link between denial-of-service, reliability and degradation of anonymity in mix networks [4]. That said, while the adversary may control a large portion of the participating entities, we assume that there are still more honest nodes than malicious nodes[2].

The adversary we consider acts as a *global passive observer*, and observes all internal states and keys of malicious nodes. Furthermore, it may launch active attacks by controlling all inputs and outputs of malicious nodes and create all functions of their secret keys. However, the adversary is limited in terms of active network attacks: we assume that they cannot drop packets between honest parties, or 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 [25] that assumes honest nodes can eventually communicate. It allows for more efficient byzantine fault tolerance schemes, such as the one we present.

We assume there are n mixes, f of which are malicious and h are honest ($n = f + h$). We refer to cascades where all mixes are malicious as *fully-malicious* cascades, to cascades where all mixes are honest as *fully-honest* cascades, and to cascades where only some of the mixes are honest as *semi-honest* cascades. Similarly, we assume there are n_d directory authorities, m_d of which are malicious and h_d are honest ($n_d = m_d + h_d$).

2.3 Cryptographic Primitives

For message encoding and decoding we assume that clients and mixes use the well-established mix packet format Sphinx [7]. Sphinx ensures all encoded and relayed messages are indistinguishable from each other, and also allows the sender to confidentially relay arbitrary information to intermediate mixes and the recipient of the message. We denote the encoding of a message using the mix format as Pack($path$; $routing$; $message$; rnd), where $path$ denotes the addresses and public keys of all mixes in the path (cascade), $routing$ includes any additional routing information the client wishes to relay to each intermediary node; $message$ is the final message that the recipient gets; and rnd is a random string of bits that are used for all calls to the random number generator relied upon by the packet format. Sphinx ensures all messages encoded are indistinguishable from each other to an adversary until they are fully processed and output by the cascade. The Sphinx packet format is used end-to-end from the sender of the message to the recipient performing the last stage of sphinx decoding. We do not rely on the reply-block facility of the format, and all messages use the forward path constructions.

We assume a PKI providing an authoritative mapping between mixes and their public keys. We also use a secure signature function $Sign(\cdot)$, with a matching verification function $Verify(\cdot)$, to exist. Although *Sign* and *Verify* use the relevant cryptographic keys in their operation, we abuse notations and write them without the keys, for simplicity.

We also use a 'keyless hash function' $H(\cdot)$; more precisely, the property we require of $H(\cdot)$ is of a pseudo-random generator (PRG).

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

2.4 Goals

The key goals of Miranda relate to alleviating and discouraging active attacks on mix networks that may have an impact on anonymity, and may facilitate active traffic analysis. This is achieved through mixes detecting unreliability and directory authorities taking action to adjust the topology of the network between epochs to marginalize actively malicious nodes. Specifically:

Goal 1. *Every active attacks, by a corrupt mix, is detected with non-negligible probability by at least one honest mix and/or directory.*

Goal 2. *Every active attack by a rogue mix, results, with non-negligible probability, with removal of at least one link connected to the rogue mix, or even removal of the rogue mix itself, from the graph of available links and mixes from which the directories choose cascades, from the next epoch.*

Goal 3. *Repeated application of Miranda lowers the overall prevalence and impact of active attacks by corrupt mixes across epochs, and limit the ability of the adversary to drop packets.*

3 MIRANDA'S DESIGN

3.1 Mix Operation

The mix networks we consider, operate in synchronous batches [15] and in rounds. Each mix receives packets within a time slot, denoted by a round number r. We include in the routing information sent within each packet to each mix along the cascade, the exact round number during which the mix should receive the packet, and the exact round number during which the mix should forward the packet. The packets are decoded by each mix, shuffled randomly, and forwarded to the next mix. Depending on the path constraints the topology of the mixes may be used in separate cascades, or in a Stratified network [15]: in the first case, each mix only is part of at most one cascade at any given epoch, whereas in the second case, each mix may be part of one or more cascades, typically at different positions. In all cases, the topology of the network is determined centrally by a set of directory authorities.

In Miranda, each mix along a cascade provides a receipt upon receiving a packet, to the sender - a client or the preceding mix along the cascade; for simplicity, we assume that the receipt is sent and received within the same round in which the packet was sent. Mixes record those receipts for packets sent to other mixes. Receipts are digitally signed [23] statements containing information about the packet p, in the form of $receipt = Sign(p \,||\, receivedflag = 1)$. Note that one could reduce the computational overhead related with public key signature and verification operations, by signing together multiple receipts, e.g., for multiple packets sent over the same cascade, and/or by signing the root of a Merkle tree [26], and prove that received packets are included in the tree; we ignore such optimization details, which are not even that essential, as signatures are not *that* computationally expensive.

In case a receipt is not provided within the expected time slot[3], the non-responding mix, i.e., the one who didn't send a receipt, must be faulty. If the packet was sent by a mix, that mix will inform the

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

directories; note that the directories cannot identify, merely based on this message, if the fault is with the non-responding mix or if the mix who sent the complaint is actually the faulty one in an effort to discredit a honest mix; therefore, the directory authorities may only disconnect the *link* between the two mixes. We later explain how a client can also cause disconnection of a link connected to the corrupt mix, upon detecting that the mix failed to send the receipt to the client.

3.2 Loop Messages

In addition to regular messages that users send to recipients, clients also send *loop messages*. These are special messages that a sender sends to *herself* periodically through the cascade. Loop messages are encoded into layered-encrypted packets just like any other message, making them indistinguishable from "regular" messages at all stages of their route, including the last hop from the final mix in the cascade back to the sender.

To efficiently support loops we leverage Sphinx [7]. Loop messages are encoded into loop packets using Sphinx, and provide a number of properties:

- **Indinstinguishability.** Loop packets are indistinguishable from normal packets at all stages of their routing without knowledge of a special *opening* value;
- **Loop Integrity.** An *opening* value convinces everyone that has seen a packet that the underlying message was a loop and not a regular message. Furthermore, they are convinced that the loop packet was well formed throughout its path when decoded by all mixes;
- **Loop Authenticity.** Only the creator of the loop message may provide a valid opening value, and others cannot easily construct a valid opening value.
- **Loop Security.** No one may forge an opening value that is valid for a non-loop packet created by an honest sender.

To achieve the above properties, loop packets are formed by a client choosing a random bit-string K, to construct:

$$\begin{aligned} p_K = \textsc{Pack}(\quad & path' = path + [S]; \\ & routing' = routing + [Pub_S]; \\ & message = \text{“}loop\text{”}; \\ & rnd = H(K) \quad) \end{aligned}$$

where S is the address of the sender, and Pub_S the public key of the sender. Note that the packet is routed all the way back to the sender using the sender's address and public key.

The tuple $(S, K_S, path, routing, K)$ acts as the opening value. It may be used to recompute p_K as well as all intermediate packets p_K^i that mix M_i should receive and emit. We call this process *loop packet verification*. We provide quick security arguments for the security properties above, heavily relying on the cryptographic properties of the Sphinx packet format [7].

The **indinstinguishability** property ensures that two packets, one encoding a loop (packet 0) and one a regular message (packet 1), cannot be distinguished at any stage of their processing by a dishonest mix node, without knowledge of the opening value. This can be argued in two stages: first the the random number rnd_1 used in the regular packet is distributed indistinguishably from the random number $rnd_0 = H(K)$. Secondly, the message_0 is padded and encrypted in the body of the Sphinx packet and protected through a key that may only be derived from the secret behind Pub_S and K. Since the adversary does not know any of those, they cannot distinguish them. Whence the loop packet is indistinguishable from a regular packet destined to S.

Loop Integrity ensures that given an opening, the validity of the loop packet may be checked by anyone, at all stages of processing. We note that the Pack function is deterministic, and the Sphinx encoding internally recreates the packet at all stages of its route. Thus any verifier may simply re-compute the loop packet p using the opening values $(S, K_S, path, routing, K)$ and check that it yields any packet it has seen, as well as a message 'loop', valid keys and routing information.

Loop Authenticity ensures only the loop packet creator may provide an opening. This is ensured by the fact that only the creator may find a K such that $H(K)$ leads to the random seed that would construct the message (second pre-image resistance). Similarly, **Loop Security** holds since, for an arbitrary rnd in a regular packet, second pre-image resistance ensures it is hard for anyone to find a K such that $rnd = H(K)$.

3.3 Overview

In order for Alice to send messages anonymously, her client acquires the list of all currently available cascades from the directory authorities, filters out the cascades which contain mixes it does not wish to work with, and randomly picks a cascade. Through that cascade, it sends an encoded mix packet to the first mix of the chosen cascade, and in return, the first mix sends back a *receipt*, acknowledging it received the packet. Each mix decodes a successive layer of encoding and verifies the validity of the expected round t and well-formedness of the packet. Malformed packets or packets that arrive on the wrong round are simply dropped. The decrypted packet is then batched and shuffled with the rest of the decrypted packets that arrived during the current round, and at the end of the round, all packets are forwarded to their next hop. The mix expects a receipt from the next mix acknowledging received packets.

In order to deter active attacks, Alice periodically crafts and sends *Loop messages*, encapsulated in encrypted packets in the same manner as any other message. If a loop message fails to complete the loop back to Alice, she queries all mixes in the cascade for evidence whether they have received, processed and forwarded the loop packet. This allows her to isolate the cascade's problematic link which caused the packet to be dropped. Alice then reports the isolated link to the directory authorities, and receives a signed receipt on her report stating that the link will no longer be used to construct future cascades.

At the end of an epoch, directory authorities engage into the process of generating new cascades for the next epoch, based on the information that was gathered in the last epoch. Newly generated cascades do not contain links that were reported, or mixes which were involved in too many reports (more than f reports).
In the rest of the paper, we separate the discussion of Miranda into two parts:

(1) **Intra-epoch operations.** During each epoch packets are relayed in synchronous rounds; clients may confront mixes

to provide evidence they relayed loop messages; and clients provide evidence of misbehavior to the directory authorities.

(2) **Inter-epoch operations.** Between epochs authorities process all denunciations about mixes' misbehavior, which they collected from clients and derive a new set of cascades that they make available for a new epoch.

We discuss the intra-epochs operations in detail in Section 4 and inter-epoch operations in Section 5.

4 INTRA-EPOCH PROCESS

4.1 Isolating Problematic Links

In order to deter active attacks, clients periodically send loop messages through their cascades. As clients are both the generators and recipients of loop messages, they know exactly during which round the message is expected to arrive. Therefore, if a loop message fails to 'complete' the loop back to the sender, the sender initiates an *isolation* process – after all mixes were meant to process the message and record evidence about them. During isolation, the client detects and isolates the specific problematic node or link in the cascade, by querying each of the mixes to establish whether they received, and hence should have forwarded, the respective layer of the loop packet.

The querying process take place by the client proving to the mixes that a specific packet was a loop by reveling its opening value. In the absence of any attacks mixes provide a receipt for the loop packet to the previous mix; and receive a receipt for the loop packet they forward. In case of lack of response (and receipt) from a subsequent mix, honest mixes must sign and share a receipt reporting the failure. Those receipts are provided to the clients performing the isolation process, after verifying the claim that a packet corresponded to a loop message.

A mix which simply drops a loop packet after sending a receipt to the previous mix can be detected as malicious beyond doubt. The preceding mix will produce a receipt that the packet was delivered correctly, but the malicious mix will not be able to produce neither a receipt that the resulting packet was either forwarded to the next mix or the link reported as faulty. Thus, malicious mixes have no incentives to follow this simple strategy to drop messages.

Instead, malicious mixes wishing to drop a packet may lie: they either do not provide a receipt for a message received (to blame the previous mix), or create a false report that the subsequent mix never responded (to blame the subsequent mix). Under those conditions a client obtains two contradictory responses: one from a mix that claims it forwarded the packet and one from a mix that claims it did not receive that packet. These conflicting claims allow the client to report the problematic link to the directory authorities, proving that indeed a packet was dropped by *one* of the conflicting mixes – however it is hard for the client or any third parties besides the two mixes to decide which of the two is malicious. Finally, the client chooses another cascade from the available cascades and continues as before. i.e., sends messages and loop messages through the new cascade.

As an example, consider a cascade of l mixes $M_1, ..., M_l$ where M_i is malicious, and p_i denotes the subsequent encrypted packet with loop message that mix M_i received. M_i decides to perform an active attack against p_i, i.e., it either drops p_i or causes it to be dropped by M_{i+1} (e.g., by delaying the packet). Because p_i is a loop packet, the sender queries all the mixes in the cascade whether they received p_i. Mix M_i can respond only in two following ways

(1) M_i claims it did not receive p_i, thus contradicting M_{i-1}'s response.
(2) M_i claims it did receive and forward p_i, thus contradicting M_{i+1}'s response.

Due to the focus on the problematic link itself rather than on the misbehaving mix, in either case, the malicious mix loses a link. The client acquires two conflicting responses about the link between (M_{i-1}, M_i) or (M_i, M_{i+1}), and sends these conflicting responses to a directory authority. The directory authority corroborates the claim made by the client, and if found valid, records the problematic link.

The combination of packet receipts, disconnection notices, and the isolation process amplify the effect of the loop packets. It forces malicious mixes to immediately lose links when they perform active attacks, by either not responding to the previous mix or recording a disconnection notice about the subsequent mix. Failure to do so in a timely manner, leads them to being completely excluded from the system. This prevents malicious mixes from "silently" attacking the system and blame other mixes when they are tested through the isolation mechanism (Goal 2).

To illustrate it, consider the case when a malicious mix decides to drop a packet. If the malicious mix intends to blame the previous honest mix, it cannot give the previous mix a receipt for that packet; otherwise, the honest mix would be able to prove its innocence by presenting the receipt that refutes the claim it did not sent the packet to the malicious mix. Such receipt allows to completely exclude the

Algorithm 1 Intra-epoch problematic link isolation

```
procedure ISOLATE(path, routing, K, receipt)
    r_1 ← receipt
    for i ← 2 ... len(path) do
        r_i ← RECEIVED(path, routing, K)          ▷ execute at M_i
    end for
    for i ← 2 ... len(path) do
        if not r_i.receivedFlag then
            σ ← (path, routing, K, r_{i-1}, r_i)
            for j ← 1 ... (m_d + 1) do
                s ← REPORT(M_{i-1}, M_i, σ)       ▷ execute at D_i
                if VERIFY(s) then
                    break
                end if
            end for
            break
        end if
    end for
end procedure

procedure RECEIVED(path, routing, K)
    p ← PACK(path, routing, "loop", PRF_K("loop"))
    receivedFlag ← (p_i received) ? 1 : 0
    return SIGN(p || receivedFlag)
end procedure
```

malicious mix from the system. Therefore, because the malicious mix does not provide a receipt, the previous honest mix immediately disconnects from the malicious mix (later, if the dropped packet was a loop packet, isolation would reach the conclusion that the problematic link was already disconnected). On the other hand, if the malicious mix intends to blame the subsequent mix, it must disconnect from the next honest mix before isolation (which might not even occur, but the malicious mix has no way of knowing that), otherwise, if isolation would take place, the client can "ask": why didn't you disconnect from the mix who did not send you the receipt? (this proves mix maliciousness).

The mere threat of loop messages, and the isolation process, therefore forces malicious mixes to drop a link with an honest mix for each messages they wish to suppress.

4.2 Reporting a Problematic Link

When clients detect a faulty link, and wish to report it, or when mixes want to disconnect from other mixes, they need to inform the directory authorities by broadcasting this information, along with evidence, to all of them. Each directory authority receives a tuples of the form (M_1, M_2, σ), where M_1, M_2 are the two mixes connected by the link, and σ is the 'evidence' showing that (at least) one of the two mixes is faulty. The evidence σ may be either a signed statement by one of the two mixes, or two conflicting statements signed by the two mixes.

Unfortunately, directory authorities might also be malicious. Therefore, they might ignore reports and attempt to maintain as many malicious links as possible. Our naive approach to prevent that, is to inform all n_d directory authorities on every report, or at least $m_d + 1$ directory authorities, thus ensuring that at least one honest directory authority would be informed. The honest directory authority will ensure that directories would not be able to ignore reports when regenerating new cascades.

In order to propagate the evidence (M_1, M_2, σ) more efficiently, clients send the evidence to an arbitrary directory authority; this directory authority would forward the evidence to at least $m_d + 1$ directory authorities, which will collectively sign it using a shared signing-key, using a threshold signature scheme, e.g., [19, 34]. Some faulty directory authorities may not agree to sign; however, this only requires the directory authority to request other (non-faulty) authorities to sign. If the selected directory authority does not return this signed receipt to the client, the client will re-send the evidence to another directory authority (or directory authorities), repeating if necessary, thereby ensuring that all such evidences reach at least one (non-faulty) directory authority.

4.3 Refusing to cooperate

Malicious mixes might attempt to circumvent the protocol by refusing to cooperate in the isolation procedure. Allegedly, this would prevent the client from obtaining the necessary proof about the problematic link, thus, preventing the client from convincing directory authorities of the problematic link. If malicious mixes refuse to cooperate, the client contacts the directory authority and asks it to perform isolation on its behalf. The client has the necessary receipt from the first mix, proving that the packet was indeed sent to the cascade, so it can prove to the directory authority that the loop message was indeed sent. If all mixes cooperate with the directory authority, the directory authority is able to isolate the problematic link, and disconnect it. If malicious mixes do not cooperate with the directory authority, the directory authority excludes these mixes from all cascades.

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, the honest directory authorities will conclude that the mix is honest, since it will be in a position to provide a receipt for the message forwarded or a disconnection notice. However, this is wasteful to directory authorities and mixes. We note that clients do not have to be anonymous vis-a-vis directory authorities, that may record false reports and eventually exclude abusive clients.

If a first mix does not cooperate by not producing 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. This allows them to increase the probability that client would select other fully-malicious cascades instead. To avoid that, clients can force the first mix to cooperate with the help of a *witness*. A witness is just another mix that will relay the packet to the misbehaving first mix. Now, the misbehaving can no longer refuse to produce a receipt, because the packet will arrive from the witness (and not from the client), which allows to perform isolation. If the witness itself is malicious, it will also refuse to produce a receipt (otherwise, it will lose a link). In that case, the client can simply choose another witness. Besides preventing malicious mixes from excluding semi-honest cascades without losing a link, the client learns about malicious mixes, and can avoid any future cascade that contains them.

5 INTER-EPOCH PROCESS

In this section, we discuss in detail the inter-epoch operations, which take place when changing from one epoch to 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 evidences they received, and use them to agree on the set of faulty links and mixes. The evidence consists of reports of faulty links from mixes, clients or authorities performing the isolation process. To achieve this, each directory authority sends to every other directory authority all new evidence of faulty links, collected since the previous epoch, together with the collective signature by $m_d + 1$ directory authorities (sent with the receipt to the mix/client). Since each of these evidences was signed by at least one non-faulty directory authority, then surely they would now be all received – with their valid signatures – by all directory authorities, therefore, all directory authorities have exactly the same set of faulty links.

Note that only links connected to (one or two) faulty mixes are ever disconnected; hence, any mix which has more than f links disconnected, must be faulty (we assume that $m < f$), 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, based on a $\mathcal{F}_{\mathcal{M}} + 1$ quorum, the *simple malicious mix filtering* step. In section 7 we discuss a more advanced filtering technique based on community detection.

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 [19, 34]. 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

We now define the *cascades selection protocol*, allowing all directory authorities to agree on a random set of cascades to be used during the upcoming epoch. The input to this protocol, in each directory authority, includes the set of n mixes $\mathcal{M} = \{M_i\}_{i=1}^{n}$, the desired number of cascades to be generated n_c, the length of cascades l, and the sets of faulty mixes $\mathcal{F}_{\mathcal{M}} \subset \mathcal{M}$ and faulty links $\mathcal{F}_{\mathcal{L}} \subset \mathcal{M} \times \mathcal{M}$. As explained above, these inputs are the same for all (non-faulty) mixes. For simplicity, $\mathcal{M}$, n_c and l are fixed throughout the execution.

The goal of the protocol is for all directory authorities to select the same set of cascades $\mathcal{C} \subseteq \mathcal{M}^l$ for the coming epoch, where $\mathcal{C}$ is uniformly chosen from all sets of cascades of length l, limited to those which satisfy a *legitimate cascade predicate* $Legit : \mathcal{M}^l \rightarrow \{0, 1\}$. We describe several possible legitimate cascades predicates in the next subsection. For example, we usually would not permit cascades where the same mix appears multiple times, or which include faulty mixes $M \in \mathcal{F}_{\mathcal{M}}$ or faulty links $(M_1, M_2) \in \mathcal{F}_{\mathcal{L}}$.

Given a specific legitimate cascade predicate $Legit$, the protocol selects, in all directory authorities, the same set of cascades, chosen uniformly at random among all cascades satisfying $Legit$. This is somewhat challenging, considering that sampling, normally, is a random process, which is unlikely to result in exactly the same results in all directory authorities.

One way to ensuring correct sampling and common output (set of cascades), is for the set of directories to compute this (randomized) sampling process jointly, using a multi-party secure function evaluation process, e.g. [20]. However, this is a computationally-expensive process. 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 $Legit$. To ensure that the results obtained by all (honest) directory authorities are identical, it remains to ensure that they use the same random bits to the algorithm. To achieve this, while preventing the faulty directory authorities from biasing the choice of the 'random' bits, we can use any coin-tossing protocol, e.g. [3], amongst the directory authorities. 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[4].

Algorithm 2 Inter-epoch cascade generation

```
procedure GENERATE(l)
    for each M_i ∈ M do
        if |F_L ∩ {M_i × M}| ≥ f then
            F_M = F_M ∪ M_i
        end if
    end for
    M ← M \ F_M
    C ← {}
    while |C| < ξ do
        ▷ Randomly select a cascade c of l mixes from M
        if LEGIT(c) then
            C ← C ∪ c
        end if
    end while
    return C
end procedure
```

5.2 Legitimate-cascade predicates *Legit*

The cascades selection protocol can use different predicates $Legit$ to define legitimate cascades. We now outline some of these predicates. Note, that some of these predicates are independent and may be applied together, to eliminate more undesired cascades. Given a cascade $c \in \mathcal{M}^l$:

- $UniqueInCascade(c) = \{\forall M_i, M_j \in c : \ i \neq j\}$
  Each mix is used only once in a particular cascade c.
- $NonFaulty(c) = \{\forall M_i \in c : \ M_i \notin \mathcal{F}_{\mathcal{M}}\}$
  Each mix in cascade c is selected only from the set of non-faulty mixes.
- $OnlyInOneCascade(c) = \{\forall M_i \in c, \ c' \in \mathcal{C} : \ M_i \notin c'\}$
  Any two cascades should not have a common mix.
- $ValidNeighbors(c) = \{\forall M_i, M_{i+1} \in c : (M_i, M_{i+1}) \notin \mathcal{F}_{\mathcal{L}}\}$
  For each pair of directly connected mixes in cascade c, this pair should not be listed in the faulty link set $\mathcal{F}_{\mathcal{L}}$.
- $ValidNodes(c) = \{\forall M_i, M_j \in c : (M_i, M_j) \notin \mathcal{F}_{\mathcal{L}}\}$
  Any two mixes in cascade c should not be selected from the non-faulty link set.

Obviously, other predicates could also be used, but it is important to weigh the effects chosen predicates have on the system. For example, consider using a predicate that enforces a direct constraint between the chosen cascades, e.g., $OnlyInOneCascade$. This predicate prevents legitimate cascades c, c' from co-existing in $\mathcal{C}$ if they share the same mix. The positive effect of this constraint is that any semi-honest cascade $c \in \mathcal{C}$, literally prevents all fully-malicious cascades that contain common malicious mixes with c, from being included in $\mathcal{C}$. On the other hand, because of the same constraint,

[4] It seems tempting to do the coin-tossing process only once, at the beginning of the execution; however, notice that, at least in theory, this may allow the attacker to cleverly select the faults so as to manipulate the choice of cascades; as a simple way to prevent such attack, the protocol would run coin-tossing once every epoch, after all faults were collected and agreed upon.

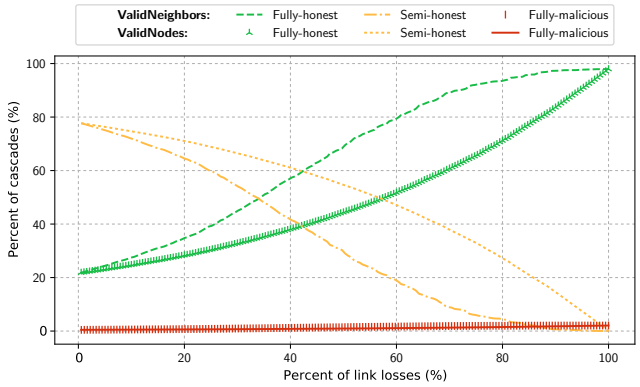

Figure 1: Probability of picking cascades as function of link losses in $ValidNeighbors$ in comparison to $ValidNodes$, where $l = 4$ and the adversary controls 30% of the mixes.

the number of possible cascades ξ in every epoch is significantly small. An adversary could take advantage of that to improve the probability of picking fully-malicious cascades during an epoch (see section 6.1). To see how too small ξ value drastically improves the abilities of the attacker, see figure 4.

Alternatively, predicates $ValidNeighbors$ and $ValidNodes$ do not prevent ξ from having a large value; therefore, they foil the feasibility of the previous attack. However, they do not limit ξ at all, and that affects both efficiency and anonymity. If mixes can appear in more than one cascade, some mixes might appear in more cascades than others; therefore, traffic probably would not distribute evenly among mixes, and moreover, there is no guarantee that all mixes would even participate in any cascade during an epoch. Furthermore, choosing too big a ξ value, will disperse users too thinly among too many cascades, which results in decreased anonymity.

Predicates also affect the *penalization factor*. Consider predicates $ValidNeighbors$ and $ValidNodes$, where a single link loss would exclude different number of cascades in each approach. In $ValidNeighbors$, 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 (see figure 1).

6 SECURITY ANALYSIS

We analyze the security of Miranda against passive attacks (subsection 6.1), and then against active attacks (subsection 6.2).

6.1 Fully-Adversarial Cascade Attacks

The use of uniform-sized (padded) packets, sent once a round by all clients, suffices to prevent traffic-analysis attacks (by an eavesdropping adversary). Furthermore, the use of a cascade of mixes, where one or more mixes in the cascade is honest, suffices to protect sender-receiver anonymity against passive attacks by the other mixes - during a single round, or provided that communication patterns do not change between rounds. Passive attackers may still perform long-term attacks based on data gathered through different rounds (e.g., intersection).

That said, even a passive adversary can easily follow packets relayed through *fully-adversarial cascades*. Consequently, such adversaries would like to divert as much traffic as possible to those fully-adversarial cascades. Attackers can try to maximize their chances to abuse the protocol, by (1) increasing the probability of fully-adversarial cascades being offered as one of the cascades in the set C which the directory authorities produce during the inter-epoch process, and/or (2) increasing the probability that users would pick a fully-adversarial cascade from C, during an epoch.

Because cascades are chosen uniformly over all valid cascades, the only way adversaries can influence cascades generation process, is by excluding non-fully-adversarial cascades. But, they can only exclude cascades by dropping links they are a part of. Hence, adversaries cannot exclude any honest links (because both mixes are honest). Moreover, they cannot exclude an honest mix from the system, because even if all adversarial mixes will disconnect from an honest mix, it's still not enough to exclude the honest mix, because we assume that $m < f$. For those reasons, adversaries cannot exclude any fully-honest cascades. Also, adversaries are unlikely to exclude adversarial links, since it will cause the loss of fully-adversarial cascades. However, adversaries are able to disconnect semi-honest cascades, by disconnecting semi-honest links, and thereby increase the probability of picking a fully-adversarial cascade.

Interestingly, we found that the attack only slightly increases the chance of selecting a fully-adversarial cascade - while significantly increasing the chance of selecting a fully-honest cascade, see Claim 1 and Figure 2. Specifically, this strategy has the following outcomes: (1) significantly improves the probability of choosing a fully-honest cascade (see Figure 3), and (2) makes it easier to detect and eliminate sets of connected adversarial domains (see Section 7).

Claim 1. *The maximum probability to pick a fully-adversarial cascade during cascades generation process, after the semi-honest cascades were excluded by the adversary is*

$$\Pr(\textit{fully adversarial}) \leq \left(\frac{m}{h-l+1}\right)^{l}.$$

Argument. Initially, the probability that a randomly selected cascade is fully-adversarial is $\Pr(\text{fully adversarial}) = \frac{\frac{m!}{(m-l)!}}{\frac{n!}{(n-l)!}} = \frac{m!(n-l)!}{n!(m-l)!}$. After the adversary disconnects all semi-honest cascades, the total number of all possible permutations of cascades is $\frac{m!}{(m-l)!} + \frac{h!}{(h-l)!}$. Thus, since each cascade is selected uniformly at random we obtain the probability of a picking fully-adversarial cascade measured as

$$\Pr(\text{fully adversarial}) = \frac{\frac{m!}{(m-l)!}}{\frac{m!}{(m-l)!} + \frac{h!}{(h-l)!}} = \left(1 + \frac{h!(m-l)!}{m!(h-l)!}\right)^{-1}$$
$$\leq \frac{m!(h-l)!}{h!(m-l)!} \leq \left(\frac{m}{h-l+1}\right)^{l}.$$

□

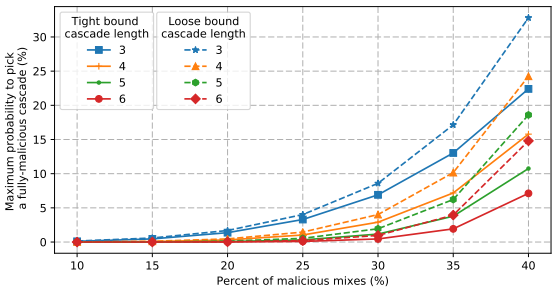

Figure 2: The maximum probability of picking a fully-adversarial cascade as a function of the cascade length and the power of the adversary.

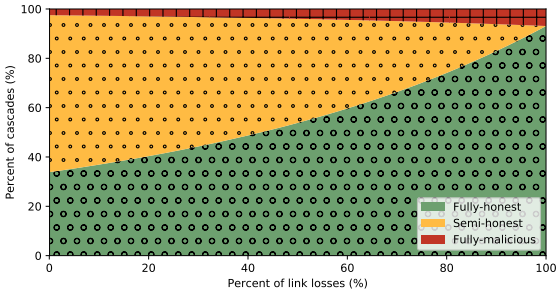

Figure 3: The probability of picking particular classes of cascades after each link loss. The parameters of the simulated mix network are $l = 3$, $n = 100$ and $m = 30$.

Once $\xi = |C|$ cascades are generated, the adversary could try to bias the probability of clients choosing a fully-adversarial cascade. To do so, the adversary can sabotage semi-honest cascades [4] through dropping messages and exclude them all. Figure 4 illustrates the number of links the adversary must affect (cost) in order to achieve a certain probability of success in shifting clients to a fully-malicious cascade. We note that the larger the number of cascades ξ, the more expensive the attack, and the lower the probability of success.

6.2 Active Attacks

In this section, we analyze how much information about the sender-recipient relationship the adversary can infer by performing an active drop attack. We first define a game modeling dropping attacks and we present the security definition and derive a bound for the adversary's advantage.

As before, we consider a network consisting of n mixes, among which m are adversarial. A population of users communicates using the Miranda infrastructure, i.e., by sending messages via the cascades of mixes.

Security Game. We define a cascade path $\mathcal{P}_x$ of length l, called *target cascade*, which we give to the adversary. As $k \geq 1$ we denote the number of compromised nodes on this path, which are controlled

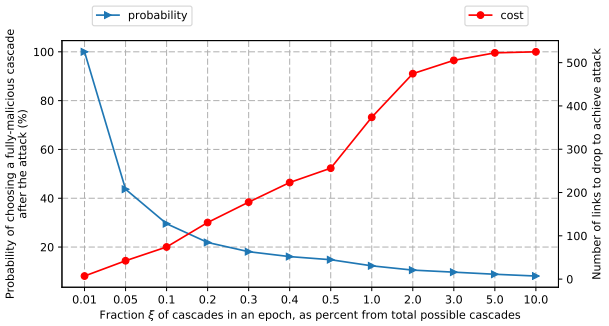

Figure 4: The cost / success ratio of performing DoS [4] attacks based on the fraction of cascades active in every epoch. Cost (red circle, right axis) is measured in links the adversary must sacrifice; probability of choosing a fully-malicious cascade (blue triangle, left axis) as a result of the attack.

by the adversary. Next, the adversary chooses two target senders A and B and two target recipients C and D. We assume that the target senders and recipients communicate using the target path $\mathcal{P}_x$. In one of the rounds the challenger $\mathcal{CH}$ selects a secret bit b at random. Depending on the value of b the following communication scenarios can happen: if $b = 0$ A sends a *challenge message* to C and B sends a *challenge message* to D, what we denote as $A \rightarrow C$ and $B \rightarrow D$, otherwise, i.e., if $b = 1$, we denote as $A \rightarrow D$ and $B \rightarrow C$.

The adversary observes the volume of traffic injected by A and B into the cascade as well as the volume of traffic received by C and D. In addition to the packets which target senders send to the target receivers, we assume that also other packets are traversing the target path, i.e., A and B are sending messages using path $\mathcal{P}_x$ to other clients in the network and also recipients C and D receive messages from other clients in the network. Since the adversary has no additional knowledge about those packets we call those packets *cover traffic*. We assume the volume of this cover traffic follows a Poisson distribution.

Let x_A, x_B denote the volume of traffic which the adversary observes sent by A and B. Similarly, let x_C, x_D denote the observed volume of traffic incoming to recipient C and D. The packets among x_C and x_D can be either the challenge packets, i.e., received from A or B or *cover packets*. Thus, let us define Y_C, Y_D as the random variables, which denote the number of cover packets received by C and D respectively, such that $Y_C \sim Pois(\lambda_C)$ and $Y_D \sim Pois(\lambda_D)$. We define $o = (x_A, x_B, x_C, x_D)$ as the observation of the volume of traffic. The goal of the adversary is to guess the value of bit b, i.e., learn which target sender is communicating with which target recipient. The adversary wins if the guess is correct.

Adversary advantage. In order to guess b, the adversary performs an active attack. Note, that we do not focus here on the fact which mix is controlled by the adversary. The position of the mix in the path does not have any impact on the adversary's success. In this active attack the adversary selects between A and B and drops a single packet injected by the selected sender.

Without loss of generality, in our analysis we assume that after A and B flush all their packets in the challenge round, the adversary

selects one of packets sent by A and drops it in one of the mixes she controls. We give the adversary additional information at the beginning of the game, meaning we tell $\mathcal{A}$ which messages sent by A and B are the challenge messages. This significantly increases the adversary's advantage, however in reality the adversary would have to guess it, which reduces the effectiveness of the attack.

Recall that in case $b = 0$ then $A \rightarrow C$ and $b = 1$ then $A \rightarrow D$. We can bound the likelihood ratio of the observation (x_C, x_D) conditioned on b, using an $\epsilon \geq 0$ and a $0 \leq \delta \leq 1$ as follows:

Claim 1. *Given an observation $O = (x_C, x_D)$ resulting from a single observation of the adversary performing a dropping attack on a single packet sent by A, the relationship of the likelihoods of the observations conditioned on the secret bit b becomes:*

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

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

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

Proof. See appendix A.

The security parameters ϵ and δ play a similar role as in differentially privacy [17] security definitions: ϵ represents the maximal amount by which the likelihood of the two events ($b = \{0, 1\}$) changes after an observation of the adversary; δ is the probability by which the leakage exceeds this ϵ. Small ϵ and δ values are better for security. We also provide a loose, but analytic, bound on δ as a function of ϵ and λ.

Claim 2. *The value of δ from Claim 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. See appendix A.2.

Evaluating the leakage (ϵ, δ). Figure 5 shows how δ behaves according to the bound (Claim 1) and the exact calculation (Claim 1), for different values of λ and a fixed leakage $\epsilon = 0.2$. As illustrated the bound is tighter for large values of λ, whereas for small values of λ one has to use a more exact calculation to obtain a good approximation of leakage.

As expected for larger volumes of cover traffic λ the values of δ become significantly lower, denoting a lesser probability of leakage. We note however that the leakage, for realistic parameters of say $\lambda = 10 \ldots 100$ is quite significant: the adversary may change their mind towards the correct b by about 20% (interpreting $\epsilon = 0.2$) and a probability of exceeding this leakage of 30%-10% ($\delta = 0.3 \ldots 0.1$). This supports the findings of previous works, presenting statistical disclosure attacks [1] and DoS based attacks [4], and arguing that

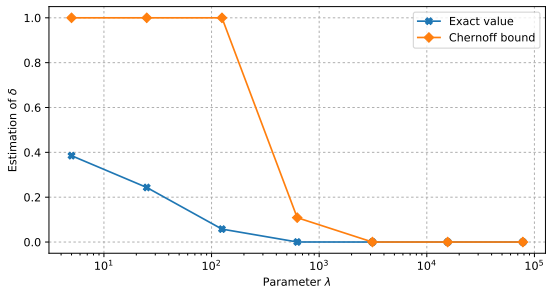

Figure 5: The precision of upper bound for δ presented in Claim 2 for a fixed $\epsilon = 0.2$. The *exact values* are computed using the importance sampling technique.

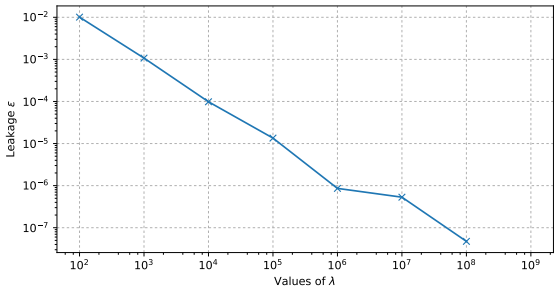

Figure 6: The comparison of amounts of leakage ϵ_∞ for different values of λ.

the traffic analysis advantage gained from dropping messages is significant. The leakage drops for larger cover traffic rates $\lambda > 10^3$ but expecting each mix client to receive over 1000 messages per round on average seems unrealistic unless large volumes of synthetic cover traffic is used.

Leakage (ϵ,δ) for multiple rounds. The advantage of the (ϵ, δ) leakage quantification presented in Claim 1 is that it composes under R multiple rounds of dropping and observations. It can be shown that after multiple rounds the likelihood ratio of the observations conditioned on b 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, those bounds leakage for multiple observations can also be shown to be tragically loose and pessimistic – since they assume that the worst case occurs in every round. In reality and adversary cannot attain such a significant advantage.

To get a better estimate of adversary's advantage we consider directly the ϵ and δ values for multiple observations. As we show in appendix A for a single observation made by the adversary the following holds:

$$\frac{Pr[Y_C = x_C, Y_D = x_D - 1 | b = 0]}{\Pr[Y_C = x_C - 1, Y_D = x_D | b = 1]} = \frac{x_D}{x_C}.$$

Using this observation we can compute the estimator $\hat{\epsilon}$ as a mean value of leakage over R observed rounds, when both $x_{C_i}, x_{D_i} > 0$.

$$e^{R\hat{\epsilon}} = \prod_{i=1}^{R} \frac{x_{D_i}}{x_{C_i}} \quad \Rightarrow \quad \hat{\epsilon} = \frac{1}{R} \sum_{i=1}^{R} \log\left(\frac{x_{D_i}}{x_{C_i}}\right),$$

for full argument see appendix B. 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 [18] we know, that as R grows, the obtained estimator tends closer to its expected value. And thus:

$$\epsilon_\infty = \lim_{R\to\infty} \hat{\epsilon} = \mathbb{E}[\log Y/X] \text{ for } X, Y \sim \text{Poisson}^+(\lambda) \tag{1}$$

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 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:

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

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 6 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 6) between the rate of cover traffic each user receives and the rate of round 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.

Two lessons can be drawn from this analysis: (1) users may wish to limit the number of correlated actions (such as sending to the same receiver), inline with previous advice relating to preventing disclosure attacks [1]; (2) Miranda should detect malicious nodes and remove them from the system, after only a small number of potential DoS attacks, to ensure the number R of potential observations remains small. The next section details how we can leverage community detection to do that.

7 APPLYING COMMUNITY DETECTION

In this section, we show that 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 used by directory authorities (see Section 5) to select cascades, by performing an additional step of filtering of nodes and propagating the faulty reports 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 set of nodes and the malicious ones, 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 [8, 10]. 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 [28].

Graph, Markov chain, and short walk definition. We augment the computations performed by directory authorities, during the inter-epoch period, by an extra step before generating new mix cascades for the subsequent epoch. We consider the graph $\mathcal{G}$ with vertices $n_i \in \mathcal{M}$ representing mixes in the system. We define an edge $(n_i, n_j) \in \mathcal{E}$ to exist between each pair of vertices if neither of n_i, n_j have been reported as faulty, and neither has the link between them been dropped by either mixes. We note that the resulting graph is symmetric, and undirected. At the beginning of time, 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 $n_i \to n_j$, that we borrow from SybilInfer [10]. We define as $\text{Deg}(n)$ the degree of the vertice n, and the probability of transiting from two vertices n_i, n_j as:

$$\Pr[n_j|n_i] = \min\left\{\frac{1}{\text{Deg}(n_i)}, \frac{1}{\text{Deg}(n_j)}\right\} \text{ if } (n_i, n_j) \in \mathcal{E} \tag{3}$$

$$\text{or } 0 \text{ otherwise.} \tag{4}$$

and call the matrix of all those transition probabilities Π, and 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 [10]. However, a short random walk, of $O(\log N)$ steps, will not converge to the stationary distribution for sparser $\mathcal{G}$ since the walks will tend to remain within regions

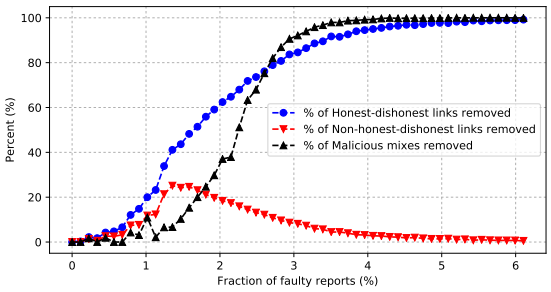

Figure 7: Effect of the community detection mechanism to detect honest-dishonest links.

of high capacitance. Similar to the insight underpinning Sybil defenses, the random walks starting from honest nodes will tend to remain within the (fully connected) regions of the graph, and the missing links between honest and malicious mixes will act as a barrier to those walks escaping in malicious regions of the graph.

We leverage this insight in the following to bias cascade construction. We define $K = \lceil k \cdot \log N \rceil$ where k is a small system constant. We then 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 a walk starting at node n_i ends up in any node n_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 n_i, we denote as cutoff $= m+1$ the smallest probability within π_i^*. Then for any node n_j such that $\pi_i^*[j] <$ cutoff the directory authorities remove the link between n_i and n_j thus further pruning the graph used to build cascades.

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 honest-malicious 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-honest-malicious 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 honest-malicious links. The results are illustrated on Figure 7, and each data point is the mean of 20 experiments – error bars are negligible.

We observe that the fraction of honest-malicious 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 honest-dishonest links; for 4% reported we prune over 90% of honest-dishonest 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. One the other hand, the fraction of non-honest-malicious 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.

Overall evaluation. It is worth contextualizing these results in terms of absolute numbers: 6% of reported faulty links – leading to nearly perfect identification of all honest-dishonest 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-honest-malicious 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 honest-dishonest links and malicious nodes.

8 CONCLUSIONS

Decryption mix networks are one of the most well-known designs for anonymous communication, due to their simple, natural design and high efficiency. However, known designs are vulnerable to subtle attacks by rogue mixes, often by selectively dropping packets. In fact, it seems that the anonymity research community has largely given up on the hope of 'fixing' layer encryption mix networks to ensure secure anonymity, and moved, instead, to more complex, less efficient designs, or to using decryption mix networks in specialized scenarios which excludes such attacks. With Miranda we show this problem is tractable, and can mitigated, at low cost.

REFERENCES

[1] Dakshi Agrawal and Dogan Kesdogan. 2003. Measuring Anonymity: The Disclosure Attack. *IEEE Security & Privacy* 1, 6 (2003), 27–34. DOI:http://dx.doi.org/10.1109/MSECP.2003.1253565

[2] Stephanie Bayer and Jens Groth. 2012. 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, Cambridge, UK, April 15-19, 2012. Proceedings*. 263–280.

[3] Mihir Bellare, Juan A. Garay, and Tal Rabin. 1996. 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)*. ACM, New York, USA, 191–200.

[4] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. 2007. Denial of service or denial of security?. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 92–102.

[5] David L Chaum. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* 24, 2 (1981), 84–90.

[6] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*. IEEE, 2–15.

[7] George Danezis and Ian Goldberg. 2009. Sphinx: A Compact and Provably Secure Mix Format. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*. 269–282.

[8] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross J. Anderson. 2005. Sybil-Resistant DHT Routing. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*. 305–318.

[9] George Danezis and Prateek Mittal. 2009. SybilInfer: Detecting Sybil Nodes using Social Networks.. In *NDSS*. San Diego, CA.

[10] George Danezis and Prateek Mittal. 2009. SybilInfer: Detecting Sybil Nodes using Social Networks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*.

[11] George Danezis and Len Sassaman. 2003. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*. ACM, 89–93.

[12] Yvo Desmedt and Kaoru Kurosawa. 2000. How to break a practical MIX and design a new one. In *Advances in Cryptology—EUROCRYPT 2000*. Springer, 557–572.

[13] Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. 2001. A Reputation System to Increase MIX-Net Reliability. In *Information Hiding, 4th International Workshop, IHW 2001, Pittsburgh, PA, USA, April 25-27, 2001, Proceedings*. 126–141.

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The second-generation onion router. In *13th USENIX Security Symposium*. Usenix.

[15] Roger Dingledine, Vitaly Shmatikov, and Paul Syverson. 2004. Synchronous batching: From cascades to free routes. In *International Workshop on Privacy Enhancing Technologies*. Springer, 186–206.

[16] Roger Dingledine and Paul F. Syverson. 2002. Reliable MIX Cascade Networks through Reputation. In *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*. 253–268.

[17] Cynthia Dwork, Aaron Roth, and others. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[18] William Feller. 1968. *An introduction to probability theory and its applications: volume I*. Vol. 3. John Wiley & Sons New York.

[19] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1996. Robust Threshold DSS Signatures. In *Advances in Cryptology—EUROCRYPT 96 (Lecture Notes in Computer Science)*, Ueli Maurer (Ed.), Vol. 1070. Springer-Verlag, 354–371.

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. 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*. New York City, 218–229.

[21] Markus Jakobsson. 1999. Flash Mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, '99Atlanta, Georgia, USA, May 3-6, 1999*. 83–89.

[22] Markus Jakobsson, Ari Juels, and Ronald L Rivest. 2002. Making mix nets robust for electronic voting by randomized partial checking.. In *USENIX security symposium*. San Francisco, USA, 339–353.

[23] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[24] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. 2014. Formal analysis of chaumian mix nets with randomized partial checking. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 343–358.

[25] Shengyun Liu, Christian Cachin, Vivien Quéma, and Marko Vukolic. 2015. XFT: practical fault tolerance beyond crashes. *CoRR, abs/1502.05831* (2015).

[26] Ralph Merkle. 2006. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87*. Springer, 369–378.

[27] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press.

[28] Abedelaziz Mohaisen, Huy Tran, Nicholas Hopper, and Yongdae Kim. 2012. On the mixing time of directed social graphs and security implications. In *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*. 36–37.

[29] Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. 2010. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 383–389.

[30] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. 2004. Mixmaster protocol – Version 2. *IETF Draft* (2004).

[31] Peter Palfrader. 2002. Echolot: a pinger for anonymous remailers. (2002).

[32] Mike Perry. 2009. Torflow: Tor network analysis. *Proc. 2nd HotPETs* (2009), 1–14.

[33] Andrei Serjantov, Roger Dingledine, and Paul Syverson. 2002. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*. Springer, 36–52.

[34] Victor Shoup. 2000. Practical Threshold Signatures. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding (Lecture Notes in Computer Science)*, Bart Preneel (Ed.), Vol. 1807. Springer, 207–220. http://dx.doi.org/10.1007/3-540-45539-6_15

[35] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, Ethan L. Miller and Steven Hand (Eds.). ACM, 137–152. http://dl.acm.org/citation.cfm?id=2815400

[36] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. 2008. Sybillimit: A near-optimal social network defense against sybil attacks. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 3–17.

A PROOFS

In this section, using the definition of the *challenge game* described in Section 6.2 we present the proof for Claim 1 and Claim 2.

A.1 Proof of Claim 1

As described in Section 6.2, without the loss of generality we consider a scenario in which the adversary targets sender A. 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., $A \to D, B \to C$. 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 $\delta = \Pr[Y_D \geq e^{\epsilon} Y_C]$. In order to calculate that we use the law of total probability and the cumulative distribution function, as presented below

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

A.2 Proof of Claim 2

In this section we present the proof of the upper bound of values δ, presented in Claim 2. The below bound can be applied to any

values of λ, however when λ is small this bound does not give as any significant information since it is a loose bound. Thus, for small values of λ the formula from Claim 1 suits better for computing values of δ.

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,

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

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

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

Now, one can notice 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

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

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

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

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,

$$\begin{aligned}Pr[Y_D \geq (1+\epsilon)Y_C] \leq & \Pr[Y_C \leq \left(1-\frac{\epsilon}{2}\right)\lambda] + \Pr[Y_C \geq \left(1+\frac{\epsilon}{2}\right)\lambda] \\ & + \Pr[Y_D \geq (1+\epsilon)\left(1-\frac{\epsilon}{2}\right)\lambda]\end{aligned} \tag{5}$$

Now by applying the Chernoff inequality [27] we can derive a final form of our upper bound for δ

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

Note, that the above bound can be made even a little bit tighter, by doing two more precise steps in equation 5. This concludes the proof.

B ESTIMATION OF THE LEAKAGE FOR MULTIPLE 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

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

From the composition theorem of differential privacy we know that given the value ϵ for a single round, the leakage after n round is computed as $n \cdot \epsilon$. However, this is the worst case , since we assume that in each round we have a worst possible observation. Hence, we focus on analysing the average case, for which we simulate several observations (x_{C_i}, x_{D_i}) and compute the estimator of the average leakage as

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

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