Low-latency mix networks for anonymous communication

Ania M. Piotrowska

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

2020

I, Ania M. Piotrowska, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

To George

Abstract

Every modern online application relies on the network layer to transfer information, which exposes the metadata associated with digital communication. These distinctive characteristics encapsulate equally meaningful information as the content of the communication itself and allow eavesdroppers to uniquely identify users and their activities. Hence, by exposing the IP addresses and by analyzing patterns of the network traffic, a malicious entity can deanonymize most online communications. While content confidentiality has made significant progress over the years, existing solutions for anonymous communication which protect the network metadata still have severe limitations, including centralization, limited security, poor scalability, and high-latency. As the importance of online privacy increases, the need to build low-latency communication systems with strong security guarantees becomes necessary. Therefore, in this thesis, we address the problem of building multi-purpose anonymous networks that protect communication privacy.

To this end, we design a novel mix network Loopix, which guarantees communication unlinkability and supports applications with various latency and bandwidth constraints. Loopix offers better security properties than any existing solution for anonymous communications while at the same time being scalable and low-latency. Furthermore, we also explore the problem of active attacks and malicious infrastructure nodes, and propose a Miranda mechanism which allows to efficiently mitigate them.

In the second part of this thesis, we show that mix networks may be used as a building block in the design of a private notification system, which enables fast and low-cost online notifications. Moreover, its privacy properties benefit from an increasing number of users, meaning that the system can scale to millions of clients at a lower cost than any alternative solution.

Impact statement

The results presented in this thesis make significant steps towards the design and study of anonymous communication systems. More specifically, this dissertation introduces novel designs for anonymous communication networks, techniques to evaluate their privacy properties, as well as mechanisms to enhance them. Moreover, the presented research has set in motion the development of open-source software and the deployment of a privacy infrastructure that allows users to control their information about their online activities.

We have designed Loopix (Chapter 3), a novel design of a low-latency anonymous communication system based on mix networks, which offers stronger security properties than the existing tools. Our design ensures metadata privacy even in the presence of powerful adversaries, yet, thanks to its scalability and performance features, is suitable for many real-time applications. Following the publication of this work in USENIX Security '17, PANORAMIX adopted Loopix as its core infrastructure design for an anonymous low-latency messaging system, the *Katzenpost* free software project.[1] Along with the project partners and developers, we released the open access specifications of the Katzenpost design and it's open-source implementation.[2,3] Furthermore, we showed how Loopix can be used to add network-level privacy for privacy focussed cryptocurrencies like Zcash in [1], which was later also added to Katzenpost [2].

We have also designed, in collaboration with the Bar-Ilan University and the University of Connecticut, the Miranda mechanism (Chapter 4), inspired during the development of Loopix, which aims to tackle a common problem in anonymous communications systems, i.e. how to detect and exclude malicious infrastructure mixes. Miranda allows the system users to detect and isolate active malicious mixes in an efficient and scalable way, without the need for computationally expensive cryptographic techniques. Our work introduces also interesting ideas for more

[1] PANORAMIX European project was a joined initiative of leading academic centres and industrial partners, whose aim was to develop an infrastructure for secure communications based on mix networks (https://panoramix.me/).

[2] https://katzenpost.mixnetworks.org/docs/specs.html

[3] https://github.com/katzenpost/

efficient detection of corrupted nodes using community detection techniques, and opens practical research questions that are likely to lead to further improvements. The result of our work was presented in USENIX Security '19.

The last technical chapter of this work introduces AnNotify (Chapter 5), a novel private notification system, which scales to millions of users at a low bandwidth and performance cost. Therefore, AnNotify can be used as an alternative to traditional PIR or privacy-preserving presence systems like DP5. This work was a result of collaboration with Bar-Ilan University and was presented in WPES '17.

The work presented in this thesis and development of Katzenpost is now further adopted and continued by the Nym Technologies company.[4] As part of Nym Technologies team, I have a chance to apply my academic and industrial experience by contributing to both research and development of the decentralized and incentivized privacy-enhancing infrastructure.

[4] https://nymtech.net/

Acknowledgements

This PhD has been a truly life-changing experience and it would not have been possible to do without the support and guidance that I received from many people.

First of all, I would like to thank my supervisor George Danezis for his time and guidance over the past years, and all of his advice. I am especially thankful for always encouraging me to explore new challenges, even those outside my comfort zone, and for teaching me to be an independent professional.

I would also like to extend my gratitude to Sarah Meiklejohn and Steven Murdoch, for their guidance and advice which helped me to significantly improve my research, and to my viva examiners, Emiliano De Cristofaro and Paul Syverson, for their precious feedback on my dissertation.

I want to also thank my previous teachers, in particular, Marek Klonowski and Michal Morayne, for giving me the opportunity to join great research projects, having confidence in me, sharing research knowledge and for encouraging me to pursue the PhD path.

Throughout my PhD years, I was very fortunate to work with amazing researchers and industrial partners. In particular, I would like to thank those who have spent some of their valuable time collaborating with me: Tariq Elahi, Jamie Hayes, Sebastian Meiser, Hemi Leibowitz, Amir Herzberg and Nethanel Gelernter.

Thank you also to Google DeepMind and Chainalysis, who gave me an opportunity to join top-notch teams as an intern. Those internships taught me many new interesting topics and allowed me to gain a lot of experience. Thank you to everyone with whom I had an opportunity to work there for making it such an amazing experience. In particular, I would like to thank Ben Laurie for his guidance and always having time for fascinating research conversations.

I would also like to express my gratitude to a wonderful group of colleagues from the InfoSec group, who had influenced my research and made my everyday work at UCL so pleasant. Thank you also to all my friends in London and beyond, for their friendship, kind words, the time we have spent together and beautiful memories.

This PhD journey would never happen without Asia, who always has been

a role model for me. Thank you for your love, friendship and encouragement. I would also like to express my gratitude to my parents and my family, for their support and enthusiasm. Especially to my aunt Iwonka for her love, kindness and support. Thank you to my Grandma Halinka and Wanda, who never knew about this adventure but were with me every step along the way. I would never study Computer Science if not amazing men in my family - My Grandpas Mietek and Stanislaw, and my uncles Gienek, Zdzisiek, Irek and Maciek. Their passionate stories about history, science and travels encouraged me to follow my dreams. Thank you for believing in me, I hope I made you all proud.

A big thank you also to Mrs Christina and Mr Andreas, for such loving welcome in their family, and always being so supportive and enthusiastic about my endeavours.

Last but not least, I would like to thank my George, for his love, support, motivation, teaching me to never give up and for making me every day a better person. This thesis, and my entire hard work, is dedicated to him.

Contents

Abstract i

Impact statment iv

Acknowledgements vi

Contents vii

List of Figures and Tables xii

1 Introduction 1

1.1 Motivation 1
1.2 Goals 4
1.3 Contributions 5
1.4 Organisation of the thesis 7

2 Background and related works 9

2.1 Technical defnitions and measures 9
2.1.1 Information entropy 9
2.1.2 Differential privacy 10
2.1.3 Bloom filters 10
2.1.4 Poisson and Exponential distribution 11
2.1.5 Memoryless property 12
2.2 Anonymity 12
2.2.1 Traffic Analysis and Active Attacks 13
2.3 Anonymous communication systems 15
2.3.1 Early designs for decentralized anonymous communication systems 15
2.3.2 Modern mix networks and DC-Nets designs 24
2.3.3 Other decentralized anonymity systems 30
2.4 Conclusion 31

I Anonymous communication systems resistant to traffic analysis and active attacks 33

3 The Loopix Anonymity System 35
3.1 Introduction . . . 35
3.2 The system high-level overview . . . 36
3.3 System model and security goals . . . 37
3.3.1 System Setup . . . 38
3.3.2 Threat Model . . . 38
3.3.3 Security Goals . . . 39
3.4 The Loopix Architecture . . . 41
3.4.1 Message packet format . . . 41
3.4.2 Message sending and cover traffic . . . 43
3.4.3 The Poisson Mix Strategy . . . 46
3.5 Analysis of Loopix security properties . . . 48
3.5.1 Passive attack resistance . . . 48
3.5.2 Poisson mix security . . . 49
3.5.3 Active attack resistance . . . 55
3.6 End-to-End Anonymity Evaluation . . . 56
3.6.1 Sender-Receiver Third-party Unlinkability . . . 57
3.7 Performance Evaluation . . . 59
3.8 Comparison with Related Work . . . 64
3.9 Discussion . . . 67
3.10 Conclusion . . . 68

4 Detecting malicious mix nodes 71
4.1 Introduction . . . 71
4.2 Impact of Active Attacks on Anonymity . . . 73
4.2.1 Security game . . . 74
4.2.2 Measurement of adversary's advantage. . . . 74
4.3 System model and security goals . . . 80
4.3.1 General System Model . . . 81
4.3.2 Threat Model . . . 81
4.3.3 Security Goals of Miranda . . . 82
4.4 The Big Picture . . . 83
4.5 Intra-Epoch Process . . . 84
4.5.1 Message Sending . . . 85
4.5.2 Processing of Received Packets . . . 85

4.5.3 Loop Messages: Detect Stealthy Attacks 87
4.5.4 Handling missing receipts 91
4.6 Inter-Epoch Process 92
4.6.1 Filtering Faulty Mixes 92
4.6.2 Cascades Selection Protocol 94
4.7 Community-based Attacker Detection 96
4.7.1 Aggresive Pair Removal 96
4.7.2 Threshold Detection Algorithm 97
4.7.3 Community detection based on random walks 101
4.8 Analysis of Active Attacks 102
4.8.1 Resisting Active Attacks 102
4.8.2 Fully Malicious Cascades Attacks 105
4.8.3 Security of Loop Messages 107
4.9 Evaluation of Community Detection Techniques 108
4.9.1 Community detection using Threshold Detection and Virtual Pair Removal 108
4.9.2 Community detection based on random walks 110
4.10 Discussion 111
4.11 Conclusion 113

II Applications of mix networks 115

5 Private Notification Service using Mix networks 117
5.1 Introduction 117
5.2 System model and Goals 118
5.2.1 High-level overview 119
5.2.2 Security Goals 119
5.2.3 Threat Model 120
5.3 The Design of AnNotify 120
5.3.1 The AnNotify Protocols 121
5.4 Security of AnNotify 124
5.4.1 Game between the adversary and the AnNotify system . . . 124
5.4.2 The Security of AnNotify 126
5.4.3 Empirical adversary advantage 135
5.4.4 Other security arguments 136
5.5 Analytical Performance Evaluation 137
5.6 Experimental Evaluation 138
5.6.1 Implementation & Infrastructure 138

5.6.2 Performance Evaluation 139
5.6.3 Comparison to DP5 140
5.7 AnNotify Extensions 143
5.8 Applications 144
5.9 Conclusions 146

6 Conclusions and Future work **147**
6.1 Limitations and Future work 149

Bibliography **153**

List of Figures and Tables

2.1 Mix operations - decryption and mixing. 16
2.2 Mix cascade. 16
2.3 Simple DC-network paradigm. 18
2.4 Example of Tor onion routing circuit. 19
2.5 Example of a P2P network. 20

3.1 The Loopix Architecture. 42
3.2 Sending a single message between two users using the Loopix system. The dotted line depicts retrieving of messages. 44
3.3 Message storage and retrival. 45
3.4 The Poisson Mix strategy mapped to a Pool mix strategy. 46
3.5 Observation $o_{n,k,l}$ 51
3.6 Entropy versus the changing rate of the incoming traffic for different delays with mean $\frac{1}{\mu}$. In order to measure the entropy we run a simulation of traffic arriving at a single Loopix mix node. 53
3.7 Likelihood difference ε vs the delay parameter μ of mix nodes. We use $\lambda = 2$, a topology of 3 layers with 3 nodes per layer and no corruption. 59
3.8 Likelihood difference ε vs the number of layers of mix nodes with 3 mix nodes per layer. We use $\lambda = 2$, $\mu = 1$, and no corruption. . . . 60
3.9 Likelihood difference ε vs the percentage of (passively) corrupted mix nodes. We use $\lambda = 2$, $\mu = 1$ and a topology of 3 layers with 3 nodes per layer. 60
3.10 Overall bandwidth and good throughput per second for a single mix node. 62
3.11 Latency overhead of the system for rates $\lambda_P = \lambda_L = \lambda_D = 10, \lambda_M =$ 10 per minute, and no additional delay added to the messages by the senders. 63

3.12 End-to-end latency histogram measured through timing mix node loops. The latency of the message is determined by the assigned delay and fits the Gamma distribution with mean 1.93 and standard deviation 0.87. 64

4.1 The precision of upper bound for δ presented in Theorem 3 for a fixed $\varepsilon = 0.2$. 78
4.2 The comparison of amounts of leakage ε_∞ for different values of λ. . 80
4.3 High-level overview of the process of isolating malicious mixes in Miranda. 84
4.4 A diagram illustrating loop packets and isolation process. 90
4.5 An illustration of the simple malicious mix filtering (without community detection). 93
4.6 Probability of picking cascades as function of link losses, where $l = 4$ and the adversary controls 30% of the mixes. 95
4.7 An illustration of how virtually removing mixes from $\overline{\mathsf{G}}$ can expose malicious mixes. Algorithm 2 refers to the graph in 4.7b as $\overline{\mathsf{G}_1}$, since it is the same graph $\overline{\mathsf{G}}$ as in 4.7a but without M_1 and without M_1's neighbors. 97
4.8 A demonstration how Miranda's community detection can significantly improve the detection of malicious mixes using an example graph $\overline{\mathsf{G}}$ and $f = n_m + 1$. 99
4.9 The maximum probability of picking a fully malicious cascade as a function of the cascade length and the power of the adversary. 105
4.10 The probability of picking particular classes of cascades after each link loss. The parameters of the simulated mix network are $l = 3$, $n = 100$ and $n_m = 30$. 106
4.11 The costs and success probability of performing DoS [3] attacks based on the fraction of cascades active in every epoch. 107
4.12 The effect of using community detection against malicious mixes. . 109
4.13 Effect of the community detection mechanism to detect semi-honest links. 111

5.1 The AnNotify architecture. 119
5.2 The empirical adversary's advantage for a single round, averaged over 10^6 samples, as a function of the number of subscribers and the number of shards. The advantage is presented on a log scale. . . 135

5.3 AnNotify's implementation evaluation summary. The system scales perfectly for the increasing number of clients. Larger shards imply higher bandwidth and cost per client. The cost evaluation was done based on Amazon EC2 `m4.large` instances. 138
5.4 Security versus Bandwidth comparison for AnNotify and DP5/IT-PIR. 142
5.5 Security versus CPU cost comparison for AnNotify and DP5/IT-PIR. 142

Chapter 1

Introduction

1.1 Motivation

The first packet sent over a computer network more than 40 years ago inaugurated the era of a digital revolution. Since then, the network technology has attracted millions of users, like no other communication system. The rapid growth of online technologies in recent decades fundamentally transformed the way we exchange information. The easily accessible Internet, even via pocket-sized phones, and a myriad of applications and tools allow people thousands of miles apart to communicate and share information within a fraction of a second. As a result, online communication became an inevitable part of our everyday lives. While the very digital world became the main tool for information flow for millions of people, opening thousands of possibilities and making our lives more convenient, it also became an endless source of personal information about its users.

This in result has a profound impact on users' privacy. The online world changed the rules for information flow, and privacy is becoming harder to maintain as the world becomes more and more connected. Nowadays information about individuals, can be easily collected by observing patterns of our online activities. Every time we send a message, visit a website, make an online purchase, or pass a cellphone tower we leave a digital trace. Those digital footprints reveal information about our social networks, spending habits, credit scores, political and religious orientations, location, or health status, and hence provide a detailed profile of individuals. Gathered data is often used to create a better user online experience and building the users' advertisement profiles. These are later used to make suggestions about websites, online shops, apps, holiday destinations, movies or music which we might like. It also allows getting in contact with social groups which share our interests. However, even more often such data is sold between the worlds largest

corporations and state-level actors, and exploited in a way which does not enable the users to control or limit the amount of processed information about them [4, 5, 6].

The NSA documents [7, 8, 9] leaked in 2013 by Edward Snowden have sparked a worldwide debate about how governments and companies misuse the data collected from the records of our online activities. Earlier projects like Trailblazer [10], which intended to track individuals by analysing communication networks, or the Protect America Act of 2007 [11], a warrant-free wiretapping and electronic surveillance law, were argued to target only those entities which might be a threat to national security. However, the uncovered documents about the PRISM program revealed that NSA intercepts and collects massive sets of online activities and Internet metadata of US and non-US citizens, even those who are not suspects of any crimes. Similarly, in the UK, the Tempora program [12] allows extracting directly from the fibre-optic cables large amounts of global Internet data, which later are shared and analysed by GCHQ and the NSA.

An increasing number of such bombshell stories [4, 7, 13], regarding widespread electronic surveillance of private communication and illicit harvesting of personal data, resulted in a surge of private communication tools. In response to the rising awareness of the fact that our daily online activities lack privacy, many Internet users turned to services offering encryption to secure their communication. Tools like PGP [14], TLS [15], WhatsApp [16], Signal [17], OTR [18, 19] etc., which offer encrypted messaging, provide an effective way to hide the content of the communication from any unauthorized parties. However, encryption is only the first step in the efforts to achieve privacy in the online world, since it can only protect the *confidentiallity* of our communication content, but is not intended to hide the distinctive characteristics associated with the communication, the so-called *metadata*. This metadata alone contains a vast amount of sensitive information [20], including the identity of the participants, duration of the communication, time, location, frequency etc., which in result can provide a detailed profile of user's interests or associations. As research has shown that the metadata leakage undermines the confidentiality properties provided by the use of encryption [21, 22, 23, 24].

Similar privacy breach happens in the context of private cryptocurrencies, like Dash [25, 26], Zcash [27, 28, 29] or Monero [30]. Even though those coins apply advanced cryptographic techniques like zero-knowledge proofs [31, 32] or ring signatures [33, 34] to offer privacy *on chain*, the users' operations are still fully exposed to the adversary who can monitor the whole or part of the peer-to-peer network, and discover the IP addresses of users who broadcast particular transactions and so identify them and easily correlate their actions [1]. Moreover, recent research uncovered that the unsecured network layer creates opportunities for centralized third

parties to monopolize the peer network and carry out large-scale attacks [35, 36].

Even the highest-rank officials acknowledge how precious the metadata is, including General Micheal Hayden, former CIA and NSA Director, who asserted *'We kill people based on metadata'* [37] or NSA General Counsel Stewart Baker who stated *'Metadata absolutely tells you everything about somebody's life. If you have enough metadata, you don't really need content'* [38]. Therefore, in an era of pervasive network monitoring, it is crucial to protect the metadata as much as the communication content itself, in order to protect users' privacy.

In response to the danger of tracking metadata of internet communication, services like VPN (Virtual Private Network) [39, 40] gained popularity. VPNs mask the users IP address and location from the receiving service provider by relaying traffic via a protected encrypted tunnel and a centralized proxy server. Similarly, tools like Anonymizer [41], forward traffic via anonymous rotating IP addresses to provide anonymous web browsing. However, such centralized systems, even though they shield our online activities, are ineffective in the presence of a powerful network eavesdroppers, who can simply track the routed network traffic, and correlate our IP address with the websites we are visiting. Moreover, the VPN provider acts as a trusted proxy, and hence knows about all of the websites an individual is accessing. But such centralized systems can be legally compelled to reveal gathered log information about their users or belong to the same handful of companies [42], even though they appear to be run by independent corporations, therefore, in reality, the anonymization offered by such services is weak.

In contrast to single proxy VPN, Tor builds upon a decentralized network of nodes run by volunteers [43]. The connections are routed via multi-hop circuits, hence none of the nodes in the circuit has visibility on both the sender and receiver of the communication. Although Tor offers much stronger anonymity properties than VPNs, Tor can be easily defeated by a global adversary performing traffic analysis [44, 45, 46, 47, 48]. To protect users' anonymity, we need a solution which is resistant both against compulsion and traffic analysis attacks.

Therefore, in this thesis, we focus on building anonymous communication systems, which minimize the amount of information held by the system's components and protect users' anonymity even against sophisticated adversaries. Moreover, our goal is to ensure that the proposed designs can scale to millions of users and be easily integrated with various everyday applications.

1.2 Goals

The goal of this work is to propose designs for online communication systems, which ensure strong confidentiality and anonymity properties for its users, by hiding both the content and the metadata associated with the online communication, even if the system is exposed to sophisticated surveillance techniques.

It has been almost four decades since David Chaum introduced the idea of anonymous communication [49]. The research community has long recognised the significant privacy risk caused by the network metadata exposure and developed several system designs which tackle the problem of anonymous communication, including the famous Tor project [43]. However, while all those systems make great strides towards enhancing users privacy on the internet, there appear to remain an unavoidable tradeoff in building anonymous communication networks - strong anonymity can be achieved only at a cost of high performance and latency overhead or poor scalability, while low-latency anonymous communication offers much weaker security guarantees [50]. Therefore, current strong anonymous mechanisms are limited to high-latency use cases or local-area settings, and are unsuitable for many practical applications, like for example large scale instant messaging or cryptocurrencies.

Another challenge in building anonymous communication systems is their reliability and resistance to active attacks, i.e., *dropping* or *delaying* packets, by malicious infrastructure nodes. Neither mix networks nor onion routing have inherent protection against such attacks. Onion routing systems, like Tor, do not offer any resistance to such attacks, while current mix network designs which address this drawback require complex and expensive proofs of correct shuffling which come at a great cost and limiting systems assumptions.[1] Such attacks, however, induce *timing signatures* which can be used to correlate the communicating users, thus have severe repercussions for privacy and efficiency of the communication. Therefore, it is important to design mechanisms which mitigate such active attacks.

Therefore, in this thesis, we propose a design of a communication system which combines strong security guarantees of mix networks without sacrificing the performance. We also focus on designing mechanisms which detect and prevent active attacks in an efficient and performance-friendly way. Furthermore, we investigate how mix networks can be used in building privacy-enhancing technologies, starting from originally proposed anonymous emails, to instant messaging, cryptocurrencies and PIR alternatives.

Overall, the main goal of this work is to develop and study a secure anonymous

[1] Tor explicitly highlights that such attacks are out of scope of it's threat model [43].

communication system suitable for many modern applications.

1.3 Contributions

The main cotributions of this thesis can be summarized as follows:

- (Chapter 3) We develop Loopix, a new message-based anonymous communication system, that allows for a tunable tradeoff between latency and genuine and cover traffic volume to foil traffic analysis. Therefore, it combines security, performance and scalability. Loopix offers resistance against a strong, global passive adversary. Moreover, we show that it provides resistance against active attacks, such as trickling and flooding. This is in contrast to the currently deployed solutions, like VPN, Tor, or other peer-to-peer anonymity networks, which protect against adversaries that monitor only a limited part of the network. We present the Poisson-mix and provide novel theorems about its properties and ways to analyze it as a pool-mix. Poisson mixing does not require synchronized rounds, can be used for low-latency anonymous communication, and provides resistance to traffic analysis. We also present a methodology to empirically estimate the security provided by particular mix topologies and other security parameter values. We provide a full implementation of Loopix and measure its performance and scalability in a cloud hosting environment. In addition to this implementation, we also present a mix network discrete event simulator, which was developed to experimentally measure the mix networks' anonymity given different system requirements and parameters.

  Part of this work was published in 26th USENIX Security Symposium 2017 [51] and it is a joined work with George Danezis, Sebastian Meiser, Jamie Hayes and Tariq Elahi. I am the main contributor of this work and provided most of the key ideas, including design, theoretical security analysis, implementation and performance evaluation. The sections which were not published are part of an ongoing work.

- (Chapter 4) We introduce Miranda, an efficient, low-cost and scalable novel design that detects and mitigates active attacks. To protect against such attacks, we leverage the reputation and local reports of faults. The Miranda design can be integrated with other mix networks and anonymous communication designs. We extend the traditional mix packet formats for verifiability, by proposing an encoding for secure loop messages, that may be used to securely test the network for dropping attacks. Moreover, we show how Miranda can take advantage of techniques like community detection in a novel way, which further improves its effectiveness.

This work was published in 28th USENIX Security Symposium 2019 [52] and it is a joined work with Hemi Leibowitz, George Danezis and Amir Herzberg. I am the main contributor to the theoretical analysis of the impact of active attacks on systems' anonymity. The remainder of the work was shared between the co-authors.

- (Chapter 5) We propose AnNotify, a new private, timely and scalable notification system, based on anonymous communication and sharding, which guarantees relationship privacy at a low bandwidth and performance cost. We provide a rigorous security analysis of AnNotify, delivering an upper bound on the information leakage, which can be applied to systems which security is based on sharding. We also propose a number of extensions, such as generic presence and broadcast notifications, and applications, including notifications for incoming messages, in anonymous communications, updates to private cached web and Domain Name Service (DNS) queries. We present an implementation of AnNotify as a web-server, which can be scaled to millions of clients at a lower cost than alternatives.

  This work was published in Proceedings of the 2017 on Workshop on Privacy in the Electronic Society (WPES) [53] and it is a joined work with George Danezis, Nethanel Gelernter, Jamie Hayes and Amir Herzberg. I have provided most of the key design ideas and security analysis. The implementation and performance evaluation was done by Nethanel Gelernter. The remainder of the work was shared between the co-authors.

Overall, the contributions of this thesis are novel designs and analysis techniques for anonymous communication. The Loopix design (Chapter 3) is the first anonymous communication network, which combines strong security properties, scalability, and support for different applications and services. While Loopix introduces an idea to detect active trickling and flooding, the Miranda mechanism (Chapter 4) further extends this idea to allow detecting malicious nodes and eventually remove them from the network. The Miranda design can complement anonymous communication networks like Loopix to further strengthen the security they offer.

The proposed Loopix infrastructure and the Miranda mechanism provide an infrastructure for privacy-enhancing applications and services, including messaging, cryptocurrencies, microblogging, etc. However, the Loopix mix network infrastructure can also be used to build a private publisher-subscriber system, like AnNotify (Chapter 5), which is a scalable and efficient alternative to the currently existing PIR solutions.

The presented research has set in motion the development of open-source software

and the deployment of a privacy infrastructure at Nym Technologies that allows users to control their information about their online activities.[2].

1.4 Organisation of the thesis

The rest of this thesis is organised as follows. In Chapter 2 we present the technical definition of anonymity and survey several attacking techniques which aim to re-identify the system participants. Next, we review the background literature on anonymous communication systems. The remainder of this thesis is divided into two parts. Part I is dedicated to the design of modern mix network communication systems. In Chapter 3, we introduce a novel mix network design, which offers strong security properties yet achieves better performance, comparing to previous similar designs, and as a result makes mix networks feasible for both high and low latency exchange of information. We present the security analysis of the design and evaluate its performance using the implemented prototype. In Chapter 4, we focus on the common problem of anonymous communication systems, namely malicious infrastructure nodes which perform active dropping or delaying attacks. We start by measuring what impact such attacks have on the capabilities of the adversary to deanonymize network users. In order to do that, we define a security game and quantify the information leakage associated with active attacks. Next, we present an efficient and scalable mechanism to detect and penalize malicious nodes. We also propose enhancements for the basic mechanism by applying community detection techniques. Part II of this thesis focuses on applications of mix network systems. In Chapter 5 we propose a design of a private notification system, which can be used as a more efficient and scalable alternative for Private Information Retrieval schemes. The main idea behind our design is based on the combination of information sharding and anonymous channels. We also propose a technique which allows quantifying the amount of information leakage, which can be applied to various security systems based on sharding. Finally, we conclude and discuss future lines of research in Chapter 6.

[2] https://nymtech.net/

Chapter 2

Background and related works

Network layer metadata is associated with most of our daily activities and interactions in the online world. The research community has long recognised the problem of exposing metadata and the significant privacy risks it leads to. In this chapter, we outline various existing systems designed as a response to an increasing need for securing network layer information. We start by outlining technical primitives used throughout this dissertation. Next, we define *anonymity* and reviewing the popular attacking techniques against it in Section 2.2. Finally, we survey various types of systems which aim to offer meta-data private communication in Section 2.3.

2.1 Technical defnitions and measures

2.1.1 Information entropy

Definition of Shannon entropy [54]. Let X be a discrete random variable over the finite set $\mathbb{X}$ with probability mass function $p(x) = \Pr(X = x)$. The Shannon entropy $H(X)$ of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in \mathbb{X}} p(x) \log p(x). \tag{2.1}$$

Shannon's entropy is one of the fundamental concepts in information theory. Conceptually, information can be thought of as being stored in a variable and the entropy of a variable is then the amount of information contained in that variable. The *amount of information* defines how difficult it is to guess the information without having to look at the variable. Thus, entropy measures the unpredictability (i.e., randomness) of the information content. The more certain or deterministic the event is, the less information the variable contains. Hence, the increase in entropy is an

increase in uncertainty. Entropy is zero when one outcome is certain to occur, and maximum when all event outcomes are equally probable.

2.1.2 Differential privacy

Definition of Differential privacy [55]. A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ε, δ)-differentially private, where $\varepsilon \geq 0$ and $\delta \in [0,1]$, if for any database $D \in \mathbb{N}^{|\mathcal{X}|}$ and $D' \in \mathbb{N}^{|\mathcal{X}|}$, differing on at most one record, and for any possible output $S \subseteq Range(\mathcal{M})$, the following in-equation holds

$$\Pr[\mathcal{M}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(D') \in S] + \delta, \tag{2.2}$$

where the probability is taken over the randomness used by $\mathcal{M}$.

Differential privacy is a rigorous mathematical definition of privacy. An algorithm is differentially private if by looking at the output, one cannot tell whether an individual's data was included in the original dataset or not. Thus, differential privacy allows collecting information about users without comprising the privacy of an individual. ε is the maximum distance between a query on database D and D'. Thus, ε is the metric of privacy leakage at a differential change in data, while δ is the probability by which the leakage exceeds this ε.

The above definition is valid for a single query in the database and not for multiple queries. However, the composition theorem of differential privacy allows for cumulative analysis of privacy loss over sequential queries.

Composition theorem for Differential Privacy [55]. Let $\mathcal{M}_i : \mathbb{N}^{|\mathcal{X}|} \to \mathcal{R}_i$ be an $(\varepsilon_i, \delta_i)$-differentially private algorithm, where $\varepsilon_i \geq 0$, $\delta_i \in [0,1]$ and $i \in \{1,\ldots,k\}$. Then, if $\mathcal{M}^k : \mathbb{N}^{|\mathcal{X}|} \to \prod_{i=1}^{k} \mathcal{R}_i$ is defined as $\mathcal{M}^k(x) = (\mathcal{M}_1(x), \mathcal{M}_2(x), \ldots, \mathcal{M}_k(x))$, then $\mathcal{M}^k$ is $(\sum_{i=1}^{k} \varepsilon_i, \sum_{i=1}^{k} \delta_i)$-differentially private.

2.1.3 Bloom filters

Bloom filters [56] are space-efficient probabilistic data structures used for representing set membership. A Bloom filter is represented as a bit vector of ℓ bits, all initially set to 0. Bloom filters do not store the elements themselves, but rather the information whether the element is in the considered set S or not. In order to do that, a Bloom filter uses k independent hash function $H_1, \ldots H_k$, such that $H_i : S \to \{0, \ldots \ell - 1\}$, (i.e., hashes each element $x_i \in S$ to one of the l array positions).

Adding elements to a Bloom filter. To add element x into a Bloom filter we calculate each $H_i(x)$, and set the corresponding bits in the filter to 1. A particular bit can be set to 1 multiple times, but only the first change has an effect.

Checking for an element in a Bloom filter. To check if an element x is in the set S represented by the Bloom filter, we check whether all positions $H_i(x)$ are set to 1. If at least one of the bits at these positions is 0, the element is definitely not in the set. If all bits are 1, then either the element is in the set, or the bits have been set to 1 during the insertion of other elements, thus resulting in a *false-positive*. Thus, a Bloom filter is a probabilistic data structure. The false-positive rate is a function of the Bloom filter's size ℓ, the number m of inserted elements, and the number k of the hash functions used [56, 57], given as

$$f(\ell,m,k) = \left(1 - e^{\frac{-km}{\ell}}\right)^k. \tag{2.3}$$

Thus, for given filter length ℓ and number of elements m the number of hash values which minimizes the false positive probability is $k = \frac{\ell}{n}\log 2$.

2.1.4 Poisson and Exponential distribution

Poisson distribution [57]. The Poisson distribution is a discrete distribution, modeling the number of times an event occurs in an interval of time. The probability mass function of a random variable $X \sim \text{Pois}(\lambda)$ is given by

$$\Pr(X = x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{2.4}$$

where $\lambda > 0$ denotes the expected value of X (i.e., the mean of the distribution), as well as the variance of X.

Exponential distribution [57]. The Exponential distribution is a continuous probability distribution modeling the waiting time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate. The probability mass function of a random variable $X \sim \text{Exp}(\mu)$ is given by

$$Pr(X = x) = \begin{cases} \mu e^{-\mu x}, & x \leq 0 \\ 0 & x < 0 \end{cases} \tag{2.5}$$

where μ is the *rate parameter* of the distribution. The expected value of X is given by $E[X] = \frac{1}{\mu}$.

The Poisson distribution provides a description of the number of occurrences per interval of time, while the exponential distribution provides description of the length of waiting time between occurrences. Thus, if the waiting times between the events are independently exponentially distributed with parameter μ, then the number of events in one unit of time has a Poisson distribution with parameters $\lambda = \mu$.

2.1.5 Memoryless property

Memorylessness is a property of a probability distribution, which means that the past information does not give us any useful information about what will happen in the future.

Definition of Memoryless property [57]. Let X be a random variable. The probability distribution of X is memoryless if for any non-negative s and t, we have

$$\Pr[X > s+t | X > t] = \Pr[X > s].$$

The memoryless property says that the probability of exceeding $s+t$ given t is the same as the probability of originally exceeding s regardless of t. The exponential and geometric distributions are the only probability functions that have the memoryless property. Thus, since both of those distributions describe the waiting time between events, it means that the waiting time until the next event does not depend on how much time has elapsed already. Thus, the past has no bearing on the future.

2.2 Anonymity

Anonymity in the digital world can be defined as a property that allows users to hide their relations to particular operations and maintain an indistinguishable identity among other users performing similar actions. Such a set of users was defined by Pfitzmann and Hansen [58] as an *anonymity set*. The larger the anonymity set, the stronger the anonymity properties of the system. In the context of *anonymous communication* the main goal is to hide the correspondence between the messages and their senders or recipients, hence in result the information *who is communicating with whom*, in various use-cases, including messaging, commerce or web browsing.

The *anonymity set* defined by Pfitzmann and Hansen is a common measure of anonymity. However, how do we measure how large is the anonymity set? In [59] the anonymity set size was measured as $\log_2 U$, where U is the number of users in the system. However, the first limitation of this approach is that the total number of

system users might not be known. Moreover, an adversary observing the mix nodes for a while may assign different probabilities for each outgoing packet being linked to the observed incoming packet. Different probabilities of different members of the anonymity set reveal a lot of information to the attacker.

Therefore, the metrics proposed by [60] and [61] introduce the *information theoretic anonymity metrics* which applies the concept of Shannon entropy (see Section 2.1.1), and thus allows us to reason about the information contained in the probability distribution, thus takes into account the observed interactions between flowing packets, and measures of the *effective* anonymity set size.

The above metric takes into account how the packets interact with each other within the network, however, it does not consider any a priori knowledge an adversary might have. Thus, in this work, we also use another metric to reason about the anonymity, the *sender-receiver third-party unlinkability*, which quantifies the expected difference in the likelihood that a message leaving the last mix node is sent from one sender in comparison to another sender. This metric measures the chances of the adversary to correctly correlate the communicating users if the adversary has an *a priori* knowledge about the potentially communicating parties, i.e., already knows that either Alice or Bob are communicating with Eva.

We apply both the *information theoretic anonymity metrics* and the *sender-receiver third-party unlinkability* to measure the anonymity of the Loopix anonymity system presented in Chapter 3.

2.2.1 Traffic Analysis and Active Attacks

The main objective of an attacker of an anonymous communication system is to link the entities exchanging information. Depending on the adversary's capabilities to control the network, we distinguish between two types of attacks: (1) *passive attacks* and (2) *active attacks*.

Passive attacks

Passive attacks typically mean that the adversary is able to observe some or all network traffic on the communication links, but does not interfere with the communication. Such adversary performing traffic analysis exploits the intercepted network traffic in order to extract metadata information from the communication patterns, including timing, size of the messages, frequency, the volume of traffic, identities of the communicating parties etc. If the adversary is able to observe the entire network infrastructure we refer to it as a *global passive adversary*. Traffic analysis attacks have become more and more sophisticated over the years. The research commu-

nity has investigated the threat of traffic analysis from different perspectives, and fundamental examples of such attack techniques include:

Intersection attacks [59] - are based on the observation that users usually repeatedly communicate within small groups of contacts. Hence, if the adversary can observe multiple rounds of communication from one sender to the same set of receivers, she can infer with whom the user is communicating by intersecting the anonymity sets of each of the sent messages.

Statistical disclosure attacks [62, 63] - assume a model, in which the target sender communicates with a fixed set of contacts, while the other users communicate uniformly at random. By observing the sending patterns of the target user and using statistical models, the adversary can infer with whom the target sender communicates.

Traffic confirmation attacks [64] - rely on the observations of the ingress and egress edges of the mix network, where the packets are injected and received. The adversary next tries to correlate the incoming and outgoing streams of traffic, in order to reveal the communicating parties.

Website fingerprinting attack [65, 66] - in which the adversary observing the traffic creates *fingerprints* of streams of traffic, based on packets size, traffic volume or timing. Such fingerprints can be later used to compare them with the traffic generated by the user.

Active attacks

In addition to the eavesdropping capabilities, active attacks entail the possibility to actively inject, drop, delay and alter traffic packets on any link in the network. Active attacks have severe repercussions for privacy, reliability and efficiency of the system and significantly increase the adversary's chances to correctly de-anonymize users. Such attacks include for example:

Tagging attacks [67] - are performed by an active adversary that can *tag* (alter), a target message entering the network, or reinject previously seen message with a tag, in order to be able to trace it when it is forwarded to the recipient, and as a result, uncover the communicating parties.

***(n-1)* attacks [68, 69]** - in which the adversary floods an honest mix node with fake messages alongside a single target message. The remaining genuine messages are dropped or delayed. This allows the adversary to distinguish the target message and trace it.

Denial-of-service attacks [3, 70] - in which the attacker disrupts the service, in order to decrease the reliability of the system, and force the retransmission of messages and hence present more opportunities for attack.

In addition to the passive and active attacks on the communication links, if any of the system components are corrupted or malicious, we assume the full collusion between all adversarial parties and that the adversary has access to all of their internal states and operations.

2.3 Anonymous communication systems

The research on anonymous networking sparked in 1981 when David Chaum [49] pioneered the idea of a *mix network*, a decentralized network of relays which allows for unlinkable internet communication. The research community has since built and analysed many new and improved designs and investigated a variety of new attacks. In this section, we first revisit the designs of early anonymous communication systems and their limitations, which allowed the privacy community to better understand the needs and challenges of such systems. Further, we outline the modern designs of such systems and their contributions.

2.3.1 Early designs for decentralized anonymous communication systems

Mix networks

The principal concept of the Chaum's anonymous communication network is the source-routed decryption mix network in which all network traffic is relayed via a set of special servers called a *mix nodes*. Each mix is represented by its RSA public key, which the sender uses to locally encrypt traversing messages. Upon receiving a message a mix strips the encryption using a matching private key. The decryption reveals the header containing the address, where the packet should be forwarded and the encrypted payload. This ensures that any third-party observer is not able to correlate the incoming and outgoing packets based on their binary representation. Despite bitwise unlinkability, an attacker can observe and correlate the timing of encrypted packets on different links. Therefore, the mix node instead of forwarding the received packets at the first-in, first-out order batches a certain threshold of messages and shuffles the decrypted batch, following a secret permutation, before sending it out. We refer to this operation as *mixing*. Mixing ensures that the timing of the arrival and the departure of the packets from the mix node do not allow correlating them. In order to guarantee strong anonymity, even if a fraction of mix nodes are malicious, messages are relayed not by one mix, but by a sequence of

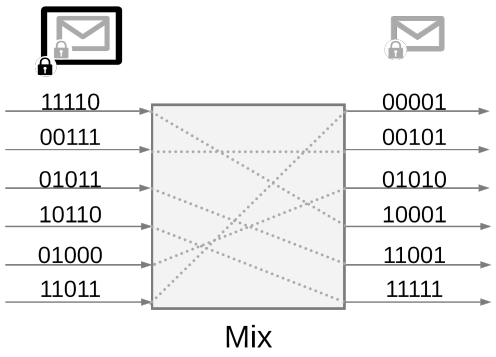


Figure 2.1: Mix operations - decryption and mixing.

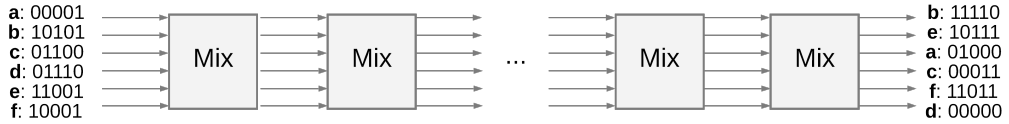


Figure 2.2: Mix cascade.

independent mix nodes, called *cascade*. The sender locally decides on the complete mix cascade that the message will traverse before reaching its final destination, and layer encrypts it using the keys of selected mixes. Thanks to that, none of the mix nodes knows the whole connection, they only know the previous and next hop, therefore only the first mix knows who is the sender, and the last mix knows who is the recipient. Hence, as long as at least a single mix in the cascade remains honest, the anonymity should be protected. Chaum's concept of a mix network is secure against a *passive adversary*, who eavesdrops part or even all the communication in the network and is not able to observe the reordering performed by honest mixes.

In addition to the *sender anonymity*, Chaum proposed the idea of *untracable return addresses* thanks to which users can not only send but also receive messages anonymously. If the user wishes to receive a response to the sent message while still keeping her identity secret, she can construct the *return address*, the same way the header of the forward message was created, and send it as part of her message. The recipient can later include this return address as the header of the response message and forward it through the network.

Even though the original design was proven to have flaws [71] in the following decades after Chaum's seminal paper many improved mix network designs were implemented by systems like Babel [72], Mixmaster [73] and Mixminion [74], designed to carry latency-tolerant communication. Mixmaster design supports sender anonymity using messages encryption but does not ensure receiver anonymity, in contrast to Babel. Following those designs, Mixminion tackles the problem of tag-

ging and reply attacks, as well as ensures forward anonymity, using link encryption between nodes, and introduces the idea of *single use reply blocks*, which support anonymous replies. As a defence against traffic analysis above designs delay incoming messages by collecting them in a pool that, if a fixed message threshold is reached, is flushed every fixed period of time, however at the cost of high-latencies. On the other side, ISDN [75], Real-time [76] and Web mixes [77] were designed to accommodate real-time, low-latency high-volume streams communication, including telecommunication (ISDN and Real-time mixes) or web browsing (Web mixes), however, those designs are only practical in the context of fixed rate traffic, such as streaming and VoIP.

The idea of traditional mix networks has evolved into a number of different designs, among others, with respect to mixing strategy. Multiple ideas of synchronous mixing techniques have been proposed, including α-mixing [78], flash mixing [79, 80], and pool mixes [81]. Further research resulted in the design of Stop-and-go mixes [82], which operate in a similar manner as classical mix nodes, but instead of batching a fixed number of messages, it delays each packet independently, hence allows operating the network in a continuous manner.

A long line of mix network research has focused on strengthening the original design against malicious mixes, which might attempt to subvert the network by dropping, modifying or duplicating packets. The literature on secure electronic elections has been preoccupied with reliable mixing to ensure the integrity of election results by using zero-knowledge proofs [83, 84, 85, 86, 87] of correct shuffling to verify that the mixing operation was performed correctly. In such designs, when a mix node shuffles the packets, it also generates a zero-knowledge proof that the outputs form a valid permutation of the input packets, while still hiding the permutation itself. However, those rely on computationally heavy primitives or require re-encryption mix networks [88], 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 networks with randomized partial checking (RPC) [89]. This cut-and-choose technique detects packet drops in both Chaumian and re-encryption mix networks, 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 [90].

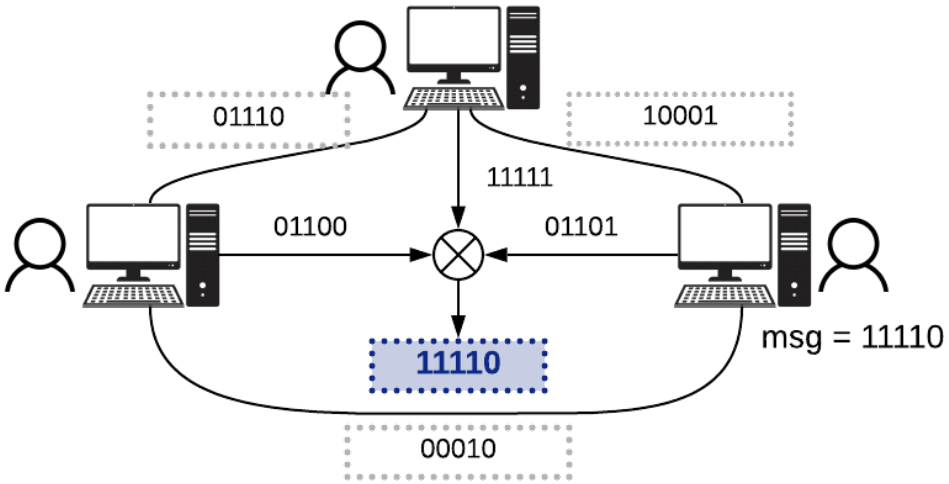


Figure 2.3: Simple DC-network paradigm.

DC-networks

Another step towards untraceable communication was the idea of the multi-party computation network called DC-nets [91], which allow a user anonymously share a message with the fellow group members. This requires each group participant to share a random binary secret with each of the remaining members. Next, each user individually XORs all the secrets they share with others, producing an obfuscated ciphertext. In order to share a message, the anonymous sender additionally XORs in the message. At the end of the round, all group members broadcast their ciphertexts among each other and next XOR together all received values. Since each binary secret is shared between exactly two participants all shared secrets cancel, revealing the message while its sender remains anonymous. Classic DC-net based designs offer strong anonymity properties but have severed practical limitations due to scalability, easy disruption and long latency. The fact that all group members have to communicate in order for the protocol to work correctly introduces significant computation and communication load on the clients. Due to network churn or disruptive clients, if any of the participants disconnects the whole protocol has to be recomputed from the beginning, hence even a single slow user delays the entire progress. Similarly, malicious participants can break the entire communication simply by broadcasting random bits. The high cost of the peer-to-peer DC-nets schemes and lack of robustness make such schemas not suitable for modern interactive communication. The later modifications of the classic protocol presented in [92], [93] offer resistance against disruption, however, all of those designs are applicable only

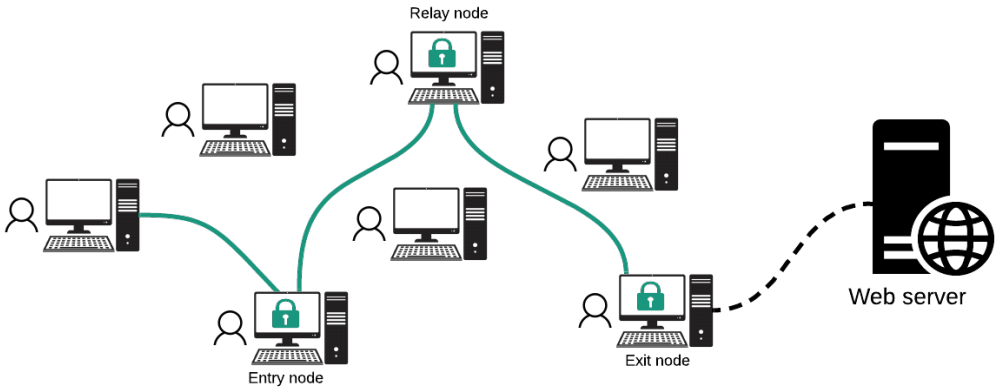


Figure 2.4: Example of Tor onion routing circuit.

in local-area settings with low delay and ample bandwidth, but not in large-scale global communication networks.

In order to improve the scalability of DC-nets Herbivore [94] divides users into many small anonymity sets, that communicate within them using traditional DC-nets. Moreover, to reduces the communication cost a single node collects and combines all users' ciphertexts. However, this approach does not allow to identify the disruptors hence is vulnerable to denial of service attacks.

Onion routing

Early designs of the mix networks or DC-nets have not found wider adoption, due to perceived higher latency that cannot accommodate real-time communications. Since mix network-based architectures have become unfashionable the idea of *onion routing* was introduced. *Onion rounting* is designed as an overlay network, based on *circuit routing*, enabling low-latency and bi-directional anonymous communication for TCP-based applications, like web browsing. Each connected user opens a *circuit*, and all communication flows down via this predetermined sequence of relays in fixed-size cells. Onion relays in the circuit share a symmetric key with the anonymous user, which key is used to layer encrypt, similarly as in mix networks, the sent packets. Upon receiving the packet, each relay decrypts one of the onion layers and forwards the packet to the next node in the circuit. In comparison to mix networks, onion routers forward the received packets in a first-in-first-out order, without altering the timing characteristics of the traffic. Not mixing the traffic however comes at a cost of privacy.

Since 2004, Tor [43], a practical manifestation of circuit-based onion routing, has become the most popular anonymous communication tool, attracting almost 2

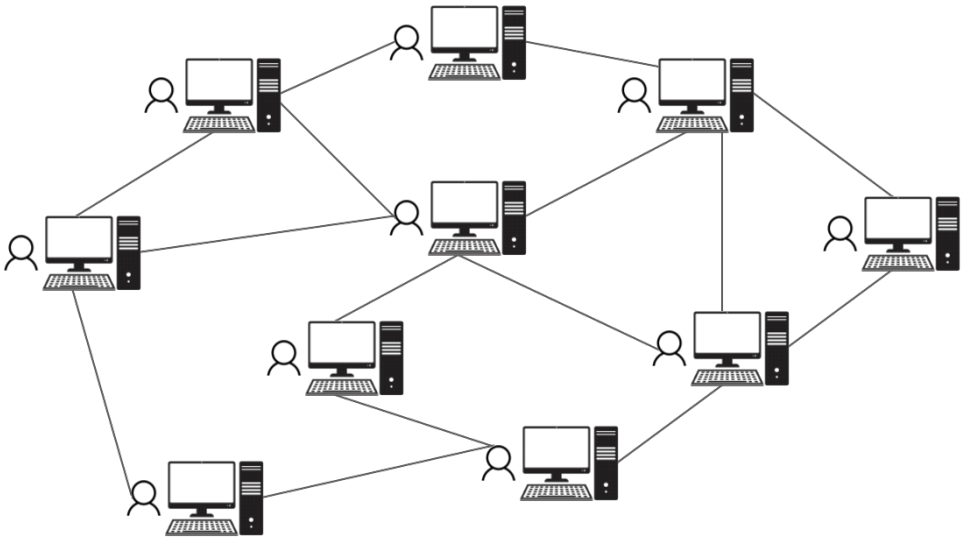


Figure 2.5: Example of a P2P network.

million users daily. Tor consists of thousands of volunteer servers, which allow it to easily scale to support more users by simply adding more peers. Unlike mix networks, onion routing systems are designed to defend against local adversaries, that can only observe a fraction of the network, modify the traffic only on this fraction, and control a fraction of the nodes. Thus, Tor's threat model is defending against websites that track the users, as well as enemies that can observe only a small part of the network, such as the user's ISP or a Tor exit node.

As research has shown, the security offered by Tor can be easily defeated by a network adversary, who can monitor both the entry and exit points of the network and perform end-to-end correlation of traffic flows [44, 45, 46, 47, 48, 95]. Moreover, even with limited capabilities, like access to the user's local connections, the adversary can identify the visited websites through website fingerprinting techniques [46, 96, 97, 98, 99]. These weaknesses of Tor exist due to the fact that even though onion encryption of the packets changes the binary representation of the flowing packets, the distinctive patterns in web traffic remains unaltered, since Tor does not add any timing obfuscation, link padding nor cover traffic. Its privacy guarantees weaken further if the adversary has capabilities to launch active attacks.

P2P Anonymous Systems

Another group of designs are P2P anonymous communication systems, in which participants act as both *end users* and *relay nodes* that route the data packets of others. P2P networks scale better than the *client-server* architecture used by mix

networks or onion routing, since new users increase the overall network capacity. However, P2P networks present also new challenges to anonymity, including the ability to locate random relays for anonymous communication. Several designs for P2P anonymous communication have been put forward, and we can broadly classify them based on their mechanism to locate peers. Designs like Crowds [100] download the complete list of participating peers from a centralized directory service, Tarzan [101], MorphMix [102] and Pisces [103] perform random walks to find other relays, while Salsa [104], AP3 [105], NISAN [106] and Torsk [107] use DHT [108] lookups to select random relays.

Crowds [100] is a system for anonymous web browsing, introduced in the late 90s. Crowds is a fully connected network which deploys *hop-by-hop* routing, meaning that the sender only selects the first relay node, which in turn picks the second node, and so on, until the message reaches its final destination. In Crowds, the nodes are grouped into the so called *crowds*, and the nodes within the same *crowd* group can communicate with each other to relay messages. Communication between peers is encrypted, however, each of intermediate peers can see the content of passing messages and the address of the final destination. Paths through the network are determined using a coin-flipping random walk technique, i.e., upon receiving a request a peer tosses a biased coin and either forwards the request to another randomly selected intermediate node, or directly to the intended recipient. The hop-by-hop routing makes Crowds vulnerable to *route capture* attacks, in which a single malicious node chooses another hostile node as its successor, thus the rest of the path will be hostile. In Crowds the *sender privacy* property is based on the uncertainty of an intermediate relay whether its predecessor is the original sender of the message or just another intermediate relay. Similarly, uncertainty on whether a relay's successor is the final destination (or just another intermediary) is what provides privacy for the message recipient. However, as research has shown, Crowds is vulnerable to the *predecessor attack* [109], which relies on the observation that clients repeatedly request particular resources, thus the corrupted node can observe how many times each of its predecessors is accessing a particular resource, and infer the peer corresponding to the most requests as the most likely initiator. Also, Crowds relies on a centralized directory service to obtain the complete list of participating peers. However, such centralized directories has serious implications, both from the perspective of privacy, availability, and scalability of the system.

Another attempt at P2P anonymous communication systems is Tarzan [101]. Similarly as in Crowds the peer initiating the traffic stream routes it via an onion encrypted circuit, however, in Tarzan the traffic is *source-routed*. In the original Tarzan design each node was required to know only a random subset of other nodes

in the network. However, as Danezis et al. [110] showed such an approach makes Tarzan vulnerable to *node profiling* attack, in which the attacker exploits the knowledge of which nodes are known to the user in order to probabilistically identify their traffic. Therefore, the final version of Tarzan requires each node to learn the identity of all peers. Tarzan employs a gossip discovery protocol to find every other peer in the network. However, this approach is believed to not scale beyond 10 000 nodes [104]. Moreover, in order to provide resilience against intermediate nodes leaving the network, Tarzan introduces a route reconstruction protocol, in which in order to rebuild a path the working peers are retained while the failed ones are replaced. However, as shown in [110] an attacker who can launch a denial of service attack against the honest nodes, could cause them fail until eventually the replacement node is one of the nodes he controlls.

In contrast to Tarzan, MorphiMix [102] requires each peer to know only a small set of other peers. Although MorphMix uses a similar architecture as Tarzan, the traffic is routed *hop-by-hop*. This type of routing is susceptible to *route capture* attacks, in which the first malicious intermediate node chooses only other malicious node to forward the traffic. In order to overcome this threat MorphMix includes the *collusion detection* mechanism that monitors whether there are any cliques in the selection of nodes in the path, thus avoids repeated connections with the same set of nodes. However, Tabriz and Borisov [111] presented an attack against the collusion detection mechanism, in which the adversary compromises a significant fraction of all anonymous tunnels.

Similar design to Tarzan and MorphMix was also proposed by Pisces [103], which leverages users' social links to build circuits for onion routing in order to considere trust relationships between the peers in the system, and thus improve the system's resilience to malicious nodes. Pisces uses random walks in the social network graph with a bias away from highly connected nodes to prevent a few nodes from dominating the circuits. Similar approch of leveraging social relationships in P2P anonymity systems was proposed also in [112] and [113], however those designs are limited in applicability to a honest-but-curious attacker model.

Another group of P2P anonymity systems are designs which rely on distributed hash tables (DHT) to obtain information about other peers in the network. DHT is a hash table, possessing key-value lookup functionality, with the index distributed among peers in a group. Each peer maintains a small routing table of its neighbours, and no single node hasthe complete list. This increases the scalability of the protocols, and minimize the number of lookups upon querying for a resource. Althought DHTs allow to avoid centralized point to manage the view of the network, they are vulnerable to various attacks on the lookup mechanism that damage the privacy and

security of the network [114]. Moreover, the partially connected network topology and the partical network view make DHT-based protocols less resilient against the DoS attacks, as well as, route fingerprinting attacks [110] and route bridging attacks [115].

An example of a system which relies on DHT lookups to locate random relays is Salsa [104]. In Salsa, the peers for the circuit are selected at random from the global pool of peers, although each node knows only a subset of nodes in the network. This allows to protect against the intersection attack and enhances scalability. However, the lookup mechanism used by Salsa is susceptible to information leak attacks [116]. Moreover, Salsa is also vulnerable to the selective DoS attacks [3].

AP3 [105] also relies on secure DHT lookups to select nodes, and uses the same routing strategy as Crowds. Similar to Salsa, the lookup mechanism used in AP3 reveals a lot of information about the lookup initiator and the communication destination, which can be used to attack the circuit construction mechanisms and compromise user anonymity [116].

In order to address the vulnerabilities of Salsa and AP3, NISAN [106] and Torsk [107] propose mechanisms specifically designed to mitigate information leak attacks. NISAN focuses on adding anonymity into the DBT lookup mechanism, while Torsk proposes the use of the so-called *secret buddy node* as a proxy to anonymize the DHT lookup initiator. However, both NISAN and Torsk were shown to be vulnerable to several passive and active attacks [117].

It is also worth mentioning other types of P2P anonymity systems. Ricochet [118] offers anonymous peer-to-peer instant messaging system that builds on Tor hidden services, in an effort to maintain its users' anonymity, however, their security properties are as good as security properties of Tor. Bitmessage [119, 120] is a peer-to-peer protocol allowing encryption of communication and metadata hiding from passive eavesdroppers, by broadcasting the encrypted messages to all users in the network. P^5 [121] also provides anonymous messaging by partitioning the peer-to-peer network into anonymizing broadcast groups. Freenet [122] offers anonymous publication and retrieval of data. Internet Invisible Project (I2P) [123] is a peer-to-peer low-latency anonymous network built on top of the Internet. I2P is designed as a closed ecosystem and does not aim to enable anonymous access to the normal Internet services. Rather than that, it is used to anonymously access hidden services integrated into the I2P network. I2P supports a variaty of applications: emails, web browsing, file sharing, blogging, forums and chatting. Moreover, the Kovri Project [124] aims to integrate I2P technology into Monero cryptocurrency to hide the IP addresses of the users interacting with the Monero blockchain. However, similarly to Tor, I2P defends only against local adversaries and is still vulnerable to

traffic analysis [125, 126, 127].

Overall, although P2P anonymous communication systems seem like a good approch, since such networks scale better than the client-server architecture used by mix networks or onion routing, such solutions present new challenges to anonymity, and are vulnerable to various attacks inherent to peer-to-peer networks, including DDoS [128], Eclipse attack [129], posioning attacks [130], Sybil attacks [131, 132] and many more.

2.3.2 Modern mix networks and DC-Nets designs

During the last decade, there were significant efforts of the research community to tackle the scalability, security and latency problems of the early designs anonymous communication systems. Modern mix network based designs achieve better scalability, lower latency, and improved reliability. In this section, we outline the most important of modern designs.

Anonymous point-to-point communication

Vuvuzela [133] is a private point-to-point metadata-hiding communication system, which goal is to combine strong privacy properties of mix networks with scalability capabilities of Tor. Vuvuzela provides resistance to a powerful adversary who can implement long term traffic analysis or block, delay and inject traffic on the communication links, but uses a weaker form of privacy known as differential privacy [134]. Vuvuzela achieves privacy through a combination of constant-bandwidth protocols, mix network and cover traffic. Vuvuzela is composed of a fixed chain of mix servers and uses the idea of *dead drops*, virtual locations hosted on the system's servers, which users use to exchange messages. Vuvuzela is composed of two protocols: (1) *dialing protocol* and (2) *conversation protocol*. The dialing protocol allows users to signal to others that they want to communicate. Hence, the users use the dialing protocol each time they want to start a conversation with a friend, even if they have talked to this friend before. In order for the dialing and conversation protocols to be secure clients and servers have to generate extensive amounts of cover traffic at a constant rate - as proposed in [133] each invitation dead drop should have a roughly equal amount of real and cover invitations/messages, which results in storing and retrieving megabytes of data. Despite reporting good scalability at an acceptable latency, those properties come at a non-trivial bandwidth cost, both for the servers and network users. Moreover, Vuvuzela does not aim to prevent servers from tampering with the users' messages, and as a

result, a malicious server can drop all but one honest user's messages.

The problem of high load and malicious servers was studied by the follow-up system Stadium [135], which investigates how to distribute workload across trustworthy servers and efficiently verify the correctness of the mixing. Stadium adopts Vuvuzela's dead-drop design and privacy goals, but instead of using traditional sequential mix cascade, where each node mixes all messages, it uses the idea of distributing the messages among parallel mix chain, where each server process only a fraction of the overall traffic. Therefore, Stadium scales horizontally hence can be deployed by incrementally adding more servers. In Vuvuzela each mix server adds the total required amount of dummy messages, therefore the total processing cost grows quadratically in the depth of the network since each server has to process all the messages added by all previous servers. In contrast, Stadium uses the collaborative noise generation, where the servers collaborate prior to mixing to inject noise messages which will be shuffled with all the traffic incoming in the round. To ensure the correctness of mixing and that the cover messages are sent along their pre-intended routes, Stadium uses the idea of *verifiable shuffling* techniques. However, those techniques come at the cost of expensive zero-knowledge proofs.

Karaoke [136] tackles this problem by using Bloom filters to efficiently check if any cover messages have been discarded: each server computes Bloom filter for all messages they receive, broadcasts the filter to all other servers and the servers check whether the cover messages they generated appear in the filter. Karaoke inherits the system architecture from Stadium and Vuvuzela, reusing the idea of dead drops and parallel chains of mix servers. To target the problem of a large volume of noise traffic that must be added in each round of previous systems, Karaoke deploys the technique of *optimistic indistinguishability*, requiring the users to always send two messages in a single round in order to simulate the appearance of an active conversation at all time. However, this approach ensures security only when there are no active attacks. An active attacker can discard user messages before they enter the system, which might reveal which two users are communicating. To protect against such scenarios Karaoke servers obscure the dead drops access patterns by generating the noise messages, each to a random dead drop and route them through a random sequence of servers. Karaoke ensures privacy with high probability by transmitting the traffic through long chains of servers, which increases the end to end latency of the communication.

AnonPop [137] is another approach towards mix network based anonymous communication system, which refines the design of Vuvuzela. Similarly like in Vuvuzela, in AnonPop all traffic is related through a sequence of synchronous mix nodes. AnonPop uses the idea of a special mailbox server, an equivalent to the *dead*

drops in [133], where users deposit and retrieve their messages. The communication with the mailbox servers is performed via a cascade of mixes. In comparison to Vuvuzela, AnonPop prevents tagging attacks, supports offline storage, and allows for efficient correspondence with multiple peers.

Similarly Pung [138], a system that aims to provide point-to-point and group anonymous messaging, uses the idea of storage mailbox servers, which mediate the exchange of messages between the communicating users, who can deposit and retrieve messages from the server. Clients upload their encrypted messages directly into the key-value mailboxes, which recipients can later access by retrieving the contents of the mailbox using computational PIR (*Private Information Retrival*) [139]. Pung hides all meta-data associated with user's conversations, even against adversaries who are capable to control all of the communication infrastructure. Hence, it withstands a stronger adversary than Vuvuzela, Stadium, or AnonPop, however at the cost of performance, which grows superlinearly with the number of users, and limited throughput, and in result suffers from high latency.

MCMix [140] is another system inspired by Vuvuzela, which provides a bidirectional anonymous messaging channel but using multiparty computation (MPC) [141]. MCMix achieves better performance than Vuvuzela, however under a weaker threat model (i.e., in a prototype with three MPC servers only one of the three can be malicious, and the system does not offer forward secrecy).

Cmix [142] is a new variant of fixed-cascade mixing network resistant to traffic analysis and intersection attacks. Cmix avoids public key operations in the real-time conversation phase by performing all computationally intensive public-key cryptographic operations in the prior precomputation phase, which does not involve users. Cmix scales linearly in the number of users and provides the sender anonymity, however, it may leak how many messages each user received and is vulnerable to tagging attacks and insider attacks [143].

Systems like Vuvuzela, Stadium or Karaoke use differential privacy to bound the information leakage on the metadata. On the other hand, systems with a stronger notion of cryptographic privacy like Pung rely on expensive cryptographic primitives and hence suffer from limited adoption. XRD [144] is an anonymous messaging system, that scales horizontally. However, so far there is no prototype implementation of such a system, which would allow comparing its overall performance with other designs. XRD scales by distributing the workload across many parallel small chains, where each chain contains at least one honest mix. Each user is assigned to a unique mailbox. When the users want to communicate they pick a number of random mix chains using a special XRD algorithm which ensures that every pair of users intersects at one of the chains. Next, users send messages to

their own mailboxes to all chosen chains, except the chain which they both picked, where instead they send messages to each other. XRD addresses the problem of active attacks, where malicious servers tamper with some of the users' messages by using the idea of aggregate hybrid shuffle, which guarantees that an honest server will either receive all honest users' messages or detect that some other previous mix server or users misbehaved. XRD has, however, two main drawbacks compared to prior system designs. With a network of N mix servers, each user has to send $O(\sqrt{N})$ messages in order to ensure that every pair of users intersects, which results in high overhead for the users and increases significantly the workload of each mix server. Hence, the cost of adding a single user to the system is expensive. Moreover, XRD is not as fast as previous designs, like Stadium or Karaoke, ($3\times$ and $25\times$ slower respectively), however, has stronger security guarantees.

Anonymous message broadcasting

The idea of DC-Net offers strong cryptographic privacy properties, however, suffers from very high computation and communication costs hence scales only to hundreds of users. This impacts the applicability but also the anonymity of the system, since the more network participants are willing to use it the better the anonymity. A new approach towards more efficient DC-Nets was introduced by Dissent [145]. Dissent combines the peer-to-peer nature of DC-Nets with the client-server solution from mix nets and onion routing. Clients are grouped and each group is assigned to a single server. Each server has a secret shared with every client. Since now clients do not share secrets with all other members of the network, but only a small fixed number of servers, the computational overhead is significantly smaller. Communication efficiency is also substantially decreased since in traditional DC-Nets each member has to transmit a quadratic amount of data. The client-server paradigm in Dissent translates to DC-net multicast trees since each client transfers data to only a single server. Servers collect the inputs from their clients, XOR them and exchange the resulting ciphertexts with other servers, and again combine all the data together. Finally, they distribute it downstream to the clients. Therefore, the communication and processing cost is linear in the number of clients. As a result, Dissent scales to thousands of clients.

Riposte [146] is another DC-Net inspired design for anonymous broadcasting of messages, suitable for latency-tolerant applications with more readers than writers, like for example Twitter or Wikileaks microblogging. Riposte offers *sender anonymity* by hiding the association between the users and their messages. In Ripost, similarly to Dissent [145], each participant sends in each epoch a fixed-length message to the system. The system architecture is composed of a small number of

independent servers, which host and maintain the shares of an anonymous bulletin board (i.e., database), which aggregates all users messages. To protect against traffic analysis attacks Riposte leverages the primitive of *write-PIR*, allowing clients to post messages on a shared bulletin board without revealing to the servers or other clients which message was posted by a particular user. In order to increase the bandwidth efficiency Riposte uses the *distributed point function* PIR technique. Instead of submitting large amounts of data to each server, the client generates a set of keys using the *distributed point function*. Next, the client sends a single key to each of the servers, which stretch those keys into vectors of the size of the database and combine them with their state of the database. This cryptographic trick allows reducing the amount of transported data between the client and each server. At the end of each epoch, the servers combine the write requests from the users, which results in a bulletin board of all client's messages. The Riposte system can be deployed using two variants. The first variant is much more computationally efficient and scales to millions of clients by using simple techniques like AES against traffic analysis and inexpensive multi-party protocol to detect and exclude malformed requests. However, this variant requires three servers such that no two of these servers collude, hence offers security under a much weaker threat model. The second variant relies on the anytrust model, where all but one server are malicious, however, is orders of magnitude much more computationally expensive since it requires for the write-PIR technique usage of elliptic curve operations and client-produced zero-knowledge proofs to protect against disruptive clients' requests. Still, even the three-server variant requires days to built an anonymity set of couple million users. Moreover, Riposte requires each client to send a message proportional to the square root of all the collection of all clients' data.

A similar approach for anonymous microblogging was proposed by Riffle [147]. Riffle offers sender and receiver anonymity in anytrust model, by combining PIR and verifiable shuffle mix network. In contrast to Riposte, Riffle uses the multiserver PIR for downloading, while the uploading protocol of the shared post consumes bandwidth proportional to only their messages rather than the number of active clients like in [145]. Riffle system consists of a small number of servers. The clients are evenly distributed among the servers, by assigning to each client his or her *primary server*. Since the techniques used for verifiable shuffle come at a large cost of expensive zero-knowledge proofs [148, 149], Riffle introduces the idea of *hybrid verifiable shuffle*, which uses classic verifiable shuffle technique to establish the symmetric keys between each client-server pair, but then uses authenticated symmetric encryption for sending messages. The verifiable shuffling of the keys guarantees that those keys originated from the legitimate clients, but

at the same time the servers cannot figure out which client sent which key thanks to the zero-knowledge property of the shuffling. If the client wants to share a new message, she onion encrypts it and uploads to a selected entry server. The server authenticates and decrypts the packets gathered during the round using the established shared keys and shuffles, before sending the result to the next server. The last server simple broadcasts all of the messages to all network users, which allows trivially to achieve receiver anonymity, however, this introduces a significant bandwidth overhead per each message. To improve the receiving efficiency Riffle replaces basic broadcast with the multi-server variant of PIR, which reduces the amount of bandwidth between servers and clients. However, even though Riffle uses the multi-server variant of PIR to improve download efficiency, each client still must perform PIR every round, even if is not interested in any message, to remain resistant to traffic analysis. Riffle protocol can be used not only for microblogging but also for file sharing. In the use-case of file sharing, Riffle achieves an order of magnitude better latency than Dissent [145] since the bandwidth overhead in [145] grows linearly with the number of clients. In terms of the microblogging, uploading a single Tweet-size post for a group of 100000 users takes on average 10 seconds, and compared to previous designs like Riposte, it achieves better performance since Riposte's bandwidth requirement for clients grows linearly with the size of the bulletin board, while the bandwidth requirement of Riffle remains the same. Although Riffle outperforms previous designs, it still leaves key challenges open. The system is not applicable to point-to-point messaging since system users can communicate only within their group, not with other network participants. Moreover, any changes in the group (e.g., clients joining or leaving) require to repeat the expensive verifiable shuffle operations.

A different approach to anonymous microblogging was presented in Atom [150]. Atom is an anonymous broadcasting schema suitable for short and latency-tolerant messages, like in microblogging or dialing applications. The principal idea of Atom is that servers are grouped into separate sets, hence each server has to process only a small fraction of the overall traffic routed through the network. Communicating users submit encrypted messages to a selected entry group. After collecting a set of messages from the users, each group collectively shuffles messages, and next divides the set into several smaller equal-size batches and forwards the re-encrypted batch to the neighbouring server. The shuffle-split-re-encrypt operations are repeated multiple times before the servers can decrypt the processed message ciphertexts and publish them on the bulletin board. Atom aims to protect against a global passive adversary performing traffic analysis. Aggregating the servers into groups allows the system to scale horizontally, meaning expanding

the network with more relays increases the overall capacity of the network. Besides being resistant to traffic analysis, Atom defends also against active attacks performed by malicious servers. However, Atom does not protect against more sophisticated attack, like for example intersection attacks [59] or Byzantine server faults [151]. Moreover, since Atom requires the shuffle operation to be repeated multiple times by many servers, this leads to the high latency of minutes or even hours, which make Atom unusable for many communication use-cases. Finally, Atom employs expensive cryptographic primitives and requires messages to be routed through hundreds of servers in series, thus incurs high latency in the order of tens of minutes for a few million users.

Aqua [152] is a high-bandwidth anonymity system which aims to provide low-latency and traffic analysis resistance for peer-to-peer file sharing applications, like BitTorrent. Aqua implements constant-transmission-rate link padding to conceal the underlying traffic patterns, and a rendezvous mechanism, which joins two sender flows at a rendezvous mix, in order to ensure anonymity for senders and receivers. Moreover, Aqua uses Tor-like infrastructure with mix nodes, where each client is attached to one entry node, which are assumed to be uncompromised. Hence, Aqua provides *k-anonymity* [153] among the k-honest clients connected to the same edge mix as long as the adversary controlling the network colludes with only one end of the path between the communicating clients. However, these assumptions result in security problems similar to Tor, since adversaries controlling both ends of the circuit can still deanonymize clients.

Herd [154], a traffic analysis resistant anonymous system for low-volume VoIP communication, uses the same technique for link padding as Aqua, however, Herd requires shorter mix paths than Aqua. It also adopts a hybrid of peer-to-peer and infrastructure based architecture, with mixes partitioned into trusted zones, and superpeers to improve scalability and uses the idea of the rendezvous mechanism to maintain the anonymity of both ends of the conversation.

2.3.3 Other decentralized anonymity systems

Another line of decentralized anonymous communication research proposed the idea of building anonymous protocols into the network layer, rather than implementing them as overlays. *Network layer anonymity systems* incorporate anonymous communication as a service of network infrastructure in the Internet and next-generation network architectures and the infrastructure routers participate in establishing the anonymous communication channels and forwarding the traffic. Rather than routing the network packets via overlay networks, network-layer anonymity

systems use direct forwarding paths. Therefore, such systems achieve much better scalability and high-throughput, however, this comes at a price of significantly degraded security guarantees in comparison to mix networks or onion routing. The first designs for network-layer anonymous systems include LAP [155] and Dovetail [156], which conceal the location of the end hosts against remote tracking by encrypting the forwarding information in the packet header. However, the binary representation of the packets does not change between the routers, hence enabling bit-pattern correlation at distinct points in the network and allowing the global adversary to easily correlate the communicating parties. Later designs, like HORNET [157] or TARANET [158] adopt ideas from onion routing and mixing in order to offer payload protection and better resistance to traffic analysis, however, both of those designs are still vulnerable to powerful attackers.

A new routing protocol called *Low Latency Anonymous Routing Protocol* (LLARP) was recently proposed by Loki [159]. LLARP is a hybrid of Tor and I2P designed exclusively for the Loki network, however, although it aims to anonymize the IP traffic, it does not attempt to solve traffic patterns correlation or nation-state network adversaries, hence it has similar security limitations as its predecessor.

Decentralized Virtual Private Networks (dVPNs), are a new VPN solution based on P2P approch. There is no central authority in dVPN, but instead each user acts as both a client and a proxy server, hence each participant offers a portion of their bandwidth to carry traffic for others. With no central point of control and failure, the dVPN system is naturally made fairer and more secure. Example of dVPNs include Orchid [160], VPN^0 [161], Sentinel [162] and Mysterium [163].

2.4 Conclusion

In this chapter, we have defined what anonymity means and gave an overview of classes of attacks against it, which have prompted a long line of research exploring how to ensure users privacy online. We outlined early designs, which pioneered the field of anonymous communication systems, and discussed in detail the modern designs. Despite all those efforts and several promising designs, the modern state-of-the-art designs of anonymous communication systems have several limitations which reduce their applicability. In the next chapters, we present a novel mix network based communication systems which aims to solve the trilemma of current anonymous networks designs: scalability, latency, reliability and security [50]. Further in this thesis, in selected chapters, we present additional overviews of other research works alongside our own work, to which it is related.

Part I

Anonymous communication systems resistant to traffic analysis and active attacks

Chapter 3

The Loopix Anonymity System

In this chapter, we introduce Loopix, a novel, scalable anonymous communication network. Loopix provides anonymous communication even against powerful global network adversaries without sacrificing the system performance, by allowing for a tunable tradeoff between latency and the volume of traffic. Importantly, an increasing number of active users in the Loopix network allows lowering the end-to-end latency and the amount of cover traffic while still offering high levels of anonymity. As a result, Loopix is the first anonymous communication network which combines strong anonymity, scalability and performance to support a large user-base and applications with various latency constraints.

3.1 Introduction

The increasing awareness of the importance of private communication motivated a renewed interest in the development of anonymous communication systems. Current state-of-the-art designs make great strides towards building private online communication systems, however, none of them can simultaneously achieve all three main goals of building a modern anonymous network, scalability, latency and security. As we have seen, systems which scale to millions of users [133, 135, 138], require large volumes of noise messages and incur perceived high latency that cannot accommodate real-time communications. On the other hand, the designs which offer low-latency [43, 152, 154] either do it at a cost of weaker anonymity or poor scalability limited to local-networks.

For this reason, we reexamine and reinvent mix-based architectures, in the form of the Loopix anonymity system. Loopix provides bi-directional anonymous communication resistant to powerful global adversaries who are capable of observing all communications and perform powerful active attacks. This is in contrast to

the currently deployed solutions, like VPN, Tor, or other peer-to-peer anonymity networks, which protect against adversaries that can monitor only a limited part of the network. Loopix offers strong security properties without sacrificing the system performance by allowing for a tunable tradeoff between latency and the volume of traffic to foil traffic analysis. Moreover, Loopix scales horizontally, hence adding more servers into the network increases its overall capacity. In comparison to previous designs, the system does not require a constant volume of cover traffic, but instead allows adjusting the amount of noise traffic depending on the deployment requirements. We demonstrate that such a mix-based architecture can support low-latency communications that can tolerate small delays, at the cost of using some extra bandwidth for cover traffic. Therefore, this system can be deployed in various applications like anonymous emails, microblogging, instant messaging or cryptocurrencies, in order to protect the anonymity at the network layer. Loopix design offers the solution for an eternal question - can we build an anonymous communication system, which combines strong security properties of mix networks with performance and scalability feasible for modern online communication.

Chapter outline: This chapter is organized as follows. In Section 3.2 we outline the high-level overview of Loopix and define our security goals and threat model. In Section 3.4, we detail the design of Loopix and describe Poisson mixes, upon which Loopix is based and introduce their properties. In Section 3.5, we present the analysis of Loopix's security properties and discuss the resistance against traffic analysis and active attacks. In Section 3.7, we discuss the implementation of Loopix and evaluate its performance. In Section 3.8, we survey related works and compare Loopix with recent designs of anonymity systems. In Section 3.9, we discuss remaining open problems and possible future work. Finally, we conclude in Section 4.11.

3.2 The system high-level overview

Loopix is a mix network-based architecture allowing *users*, distinguished as *senders* and *receivers*, to route messages anonymously to each other using an infrastructure of *mix* servers, acting as relays. Loopix follows the stratified topology [164], where mix servers are grouped into a fixed number of layers, to ensure both horizontal scalability and a sparse topology that concentrates traffic on a few links [165]. Each mix, at any given time, is assigned to one specific layer i, and connected with every mix in layers $i-1$ and $i+1$. The stratified topology has been shown optimal for anonymity, scalability and ease of analysis [166, 167]. A similar network topology

was also used in [168], however, in this design each node in layer *i* had to send one copy of every packet it receives to each node in the next layer. In Loopix packets are *source-routed*, meaning that the sender of the message selects the full route for the packet by picking a single node at random from each layer.

Loopix adopts and leverages an architecture by which users of the system are associated with service *providers* that mediate their access to the mix network, acting as the entry-exit point. Each provider has a long-term relationship with its users and may authenticate them, potentially bill them, or discontinue their access to the network. Each provider is connected to each mix in the first layer, in order to inject packets into the mix network, and also to every mix in the last layer, to receive egress packets. The provider not only serves as an access point but also acts as an always-online storage proxy, where users' incoming messages can be stored when they are offline. This architecture brings a number of potential benefits, such as resistance to Sybil attacks, enabling anonymous blacklisting [169] and payment gateways [170].

In contrast to previous anonymous messaging designs [74, 133, 146], Loopix does not operate in deterministic rounds but runs as a continuous system. This means that incoming messages can be retrieved at any time, hence users do not have to worry about lost messages when they are off-line. Additionally, Loopix uses the Poisson mixing technique that is based on the independent delaying of messages, instead of batching, which makes the timings of packets unlinkable. This approach does not require the synchronization of client-provider rounds and does not degrade the usability of the system for temporarily off-line clients.

Moreover, Loopix introduces different types of cover traffic to foil de-anonymization attacks: *loop cover traffic* and *drop cover traffic*. Mix nodes and clients self-monitor and protect against active attacks via self-injected loops of traffic. Those loop packets also serve as cover traffic to provide stronger anonymity and a measure of sender and receiver unobservability. The amount of the cover traffic used is not constant and can be adjusted depending on the system parameters and requirements, like a number of clients, volume of communication, scalability or latency demands.

3.3 System model and security goals

Before describing the Loopix system in detail, we first define the adversary model and the security goals, which our system aims to achieve.

3.3.1 System Setup

The Loopix system consists of a set of mix nodes, N, and providers, P. We consider a population of u users communicating through Loopix, each of which can act as *sender* and *receiver*, denoted by indices S_i, R_i, where $i \in \{1,\ldots,U\}$ respectively. We denote the set of all the u users as U. Each entity of the Loopix infrastructure has its unique public-private key pair (sk, pk). In order for a *sender* S_i, with a key pair (sk_{S_i}, pk_{S_i}), to send a message to a *receiver* R_j, with a key pair (sk_{R_j}, pk_{R_j}), the sender needs to know the receiver's Loopix *network location*, i.e., the IP address of the user's provider and an identifier of the user, as well as the public encryption key pk_{R_j}. We assume that this information can be made available through a privacy-friendly lookup or introduction system for initiating secure connections [171].

3.3.2 Threat Model

Loopix assumes sophisticated, strategic, and well-resourced adversaries concerned with linking users to their communications and/or their communication partner(s). As such, Loopix considers adversaries with three distinct *capabilities*, that are described next.

Global passive adversary. Firstly, a *global passive adversary* is able to observe all network traffic between users and providers and between mix servers. This adversary is able to observe the entire network infrastructure, launch network attacks such as BGP re-routing [172], or conduct indirect observations such as load monitoring and off-path attacks [173]. Thus, the GPA is an abstraction that represents many different classes of adversaries able to observe some or all information between network nodes.

Malicious infrastructure. Secondly, the adversary can observe all of the internal states of some corrupted or malicious mix relays. The adversary may inject, drop, or delay messages. She also has access to, and leverages, all secrets of those compromised parties. Furthermore, such corrupted nodes may deviate from the protocol, or inject malformed messages. A variation of this ability is where the mix relay is also the provider node meaning that the adversary additionally knows the mapping between clients and their mailboxes. When we say that a provider node is *corrupt*, we restrict that node to being honest but curious. In Loopix, we assume that a fraction of mix/provider relays can be corrupted or are operated by the adversary.

Malicious users. Finally, the adversary has the ability to participate in the Loopix system as a compromised user, who may also deviate from the protocol. We assume

	GPA	Corrupt mixes	Corrupt provider	Insider
Sender-Recipient Third-Party unobservability	✓	✓	✓	✓
Sender unobservability	✓	✓	✓	N/A
Sender anonymity	✓	✓	✓	✓
Receiver unobservability	✓	✓	✗	N/A
Receiver anonymity	✓	✓	✗	N/A

Table 3.1: The summary of Loopix's security properties against different threats.

that the adversary can control a limited number of such users—effectively excluding Sybil attacks [132] from the Loopix threat model—since we assume that *honest providers* are able to ensure that at least a large fraction of their users base are genuine users faithfully following all Loopix protocols. Thus, the fraction of users controlled by the adversary may be capped to a small known fraction of the user base. We further assume that the adversary can control a compromised user in a conversation with an honest user, and become a *conversation insider*.

3.3.3 Security Goals

The Loopix system aims to provide resistance against both passive and active attacks — including end-to-end correlation and $(n-1)$ attacks. These properties are inspired by the formal definitions from AnoA [174]. All security notions assume a strong adversary with information on all users, with up to one bit of uncertainty. We note, that AnoA framework provides a *snapshot* anonymity but does not consider long-term intersection attacks, or other attacks in which the adversary is assumed to observe the system over time [175, 176]. We start by defining the notion used throughout this thesis. Next, we define the security goals which our system aims to provide, summarized also in Table 3.1.

Notation

Throughout this thesis, we use the following notation. As $\{S \to R\}$ we denote a communication from the sender S to the receiver R, $\{S \to\}$ denotes that there is a communication from S to any receiver and $\{S \not\to\}$ denotes that there is no communication from S to any receiver, however, S may still send cover messages. Analogously, we write $\{\to R\}$ to denote that there is a communication from any sender to the receiver R and $\{\not\to R\}$ to denote that there is no communication from any sender to R, however, R may still receive cover messages.

Definition of security goals

Sender-Receiver Third-party Unlinkability. The senders and receivers should be

unlinkable by any unauthorized party. Thus, we consider an adversary that wants to infer whether two users are communicating. We define *sender-receiver third party unlinkability* as the inability of the adversary to distinguish whether $\{S_1 \to R_1, S_2 \to R_2\}$ or $\{S_1 \to R_2, S_2 \to R_1\}$ for any online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice.

Loopix provides strong sender-receiver third-party unlinkability against the GPA even in collaboration with corrupt mix nodes. We refer to Section 3.5.2 for our analysis of the unlinkability provided by individual mix nodes, Section 3.6 for a quantitative analysis of the sender-receiver third-party unlinkability of Loopix against the GPA and honest-but-curious mix nodes, and Section 3.5.3 for our discussion on malicious mixes performing active attacks.

Sender unobservability. Whether or not senders are communicating should be hidden from an unauthorized party. We define *sender unobservability* as the inability of an adversary to decide whether a specific online sender S is communicating with any receiver $\{S \to\}$ or not $\{S \not\to\}$, for any concurrently online honest sender S of the adversary's choice.

Loopix provides strong sender online unobservability against the GPA and even against a *corrupt provider*. We refer to Section 3.5.1 for our analysis of the latter.

Note, that sender online unobservability directly implies the notion of *sender anonymity* where the adversary tries to distinguish between two possible senders communicating with a target receiver. Even if the receiver is under control of the adversary, the sender's anonymity is protected. Formally, $\{S_1 \to R, S_2 \not\to\}$ or $\{S_1 \not\to, S_2 \to R\}$ for any concurrently online honest senders S_1 and S_2 and any receiver of the adversary's choice. Loopix provides sender anonymity even in light of a conversation insider, i.e., against a corrupt receiver.

Receiver unobservability. Whether or not receivers are communicating should be hidden from an unauthorized party. We define *receiver unobservability* as the inability of an adversary to decide whether any sender is communicating with a specific receiver R $\{\to R\}$ or not $\{\not\to R\}$, for any online or offline honest receiver R of the adversary's choice.

Loopix provides strong receiver unobservability against the GPA, under the condition of an *honest provider*. We show in Section 3.5.1 how an honest provider assists the receiver in hiding received messages from third party observers.

Note, that receiver unobservability directly implies the notion of *receiver anonymity* where the adversary tries to distinguish between two possible receivers in commu-

nication with a target sender. Formally, $\{S \to R_1, \not\to R_2\}$ or $\{\not\to R_1, S \to R_2\}$ for any concurrently online honest sender S and any two honest receivers R_1, R_2 of the adversary's choice. [1]

Non-Goals. Loopix provides anonymous unreliable datagram transmission and facilities replying to sent messages (through add-ons). This choice allows for flexible traffic management, cover traffic, and traffic shaping. On the downside, higher-level applications using Loopix need to take care of reliable end-to-end transmission and session management. In Section 3.9, we discuss how the follow-up works on Loopix offer reliable end-to-end transmission.

The provider-based architecture supported by Loopix aims to enable managed access to the network, anonymous blacklisting to combat abuse [169], and payments for differential access to the network [170]. However, we do not discuss these aspects of Loopix in this work and concentrate instead on the core anonymity features and security properties described above.

3.4 The Loopix Architecture

In this section, we present detailed description of the Loopix system, depicted on Figure 3.1.

3.4.1 Message packet format

All messages in Loopix are *end-to-end encrypted* and encapsulated into packets to be processed by the mix network. We use the Sphinx packet design [177], to ensure that intermediate mixes learn no additional information beyond some routing information. All messages are padded to the same length, which hides the path length and the relay position and guarantees unlinkability at each hop of the messages' journey over the network. The Sphinx packet format allows for detection of tagging attacks and replay attacks.

Each message wrapped into the Sphinx packet consists of a concatenation of two separate parts: a header, carrying the layered encryption of meta-data for each hop, and the encrypted payload, which allows for confidential message exchange. The header provides each mix server on the path with confidential meta-data, which is necessary to verify packet integrity and correctly process the packet. The structure

[1] If the receiver's provider is honest, Loopix provides a form of receiver anonymity even in light of a conversation insider: a corrupt sender that only knows the pseudonym of a receiver cannot learn which honest client of a provider is behind the pseudonym.

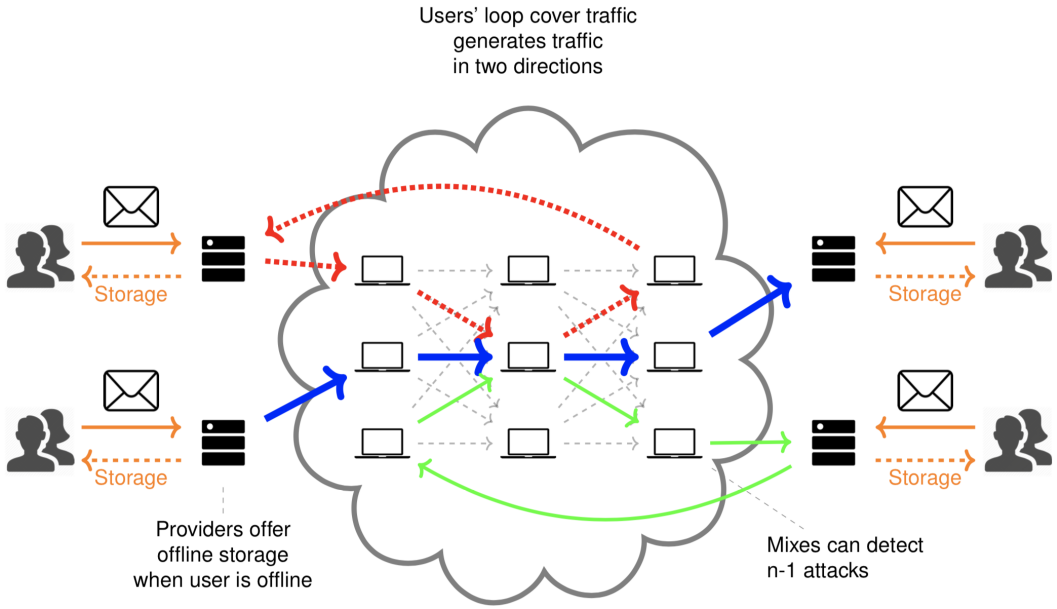


Figure 3.1: The Loopix Architecture.

of the header consists of (I) a single element of a cyclic group that is re-randomized at each hop, (II) an onion-encrypted vector, with each layer containing the routing information for one hop, and (III) the message authentication code MAC_i, which allows header integrity checking. The payload is encrypted using the LIONESS cipher [178], which guarantees that in case the adversary modifies the payload in transit, any information contained in it becomes irrecoverable. Thanks to the message authentication code in the header and the LIONESS encryption the Sphinx packet format thus allows for detection of tagging attacks.

We extend the Sphinx packet format to carry additional routing information in the header to each intermediate relay, including a delay and additional flags.

Sphinx packet generation The sender, given the public keys of the recipient and the nodes in the path, computes the sequence of shared secrets and blinded group elements. Next, the sender encrypts with the derived secret keys the vector of routing information and corresponding message authentication codes. The sender concatenates the computed header and onion-encrypted payload encapsulating confidential message to send to the recipient.

Sphinx packet processing Each node after receiving the packet proceeds as follows. First, it computes a shared key using the group element included in the packet header and its private key. Next, using the computed shared key, the node validates the integrity of the packet by computing the hash of the encrypted routing information vector and comparing it with the received MAC. If the MAC is correct, the

node, using the obtained key, strips off a single layer of encryption from the routing information and payload. The decryption operation returns the routing commands, including address of the next hop, and a new packet, which should be forwarded to the next hop.

3.4.2 Message sending and cover traffic

Path selection. In Loopix messages are routed through l layers of mix nodes, assembled in a stratified topology [164, 165]. Each mix node is connected only with all the mix nodes from adjacent layers. This ensures that few links are used, and those few links are well covered in traffic, since stratified topologies mix well in few layers [164]. As opposed to circuit-based onion routing, in Loopix the communication path for every single message is chosen independently, even between the same pair of users. Hence, each packet injected into the mix network traverses an independent path of relays, which prevents the correlation of streams of traffic. To compose a new path the client picks a single mix node at random from each layer. Providers act as the first and last layer of mix servers, hence are attached at the beginning and end of the random selected path.

Preparing message for sending. To send a message, the sender generates a random path, as described above. For each hop in the path the sender samples a random delay from an exponential distribution ($\mathsf{Exp}(\cdot)$) with parameter μ, and includes it in the vector of routing commends, together with any other auxiliary information, to the corresponding relay. Only the authrorized mix relay can reveal the routing commands encapsulated in particular encryption layer. Given the message, recipient, path and routing commends the client encapsulates them into a Sphinx packet format.

Sending messages and cover traffic. Users and mix servers continuously generate a bed of *real* and *cover traffic* that is injected into the network. Our design guarantees that all outgoing traffic sent by users can by modeled by a Poisson process.

• **Message sending.** To send a message, a user packages their message into a mix packet and places it into their *buffer*—a first-in-first-out (FIFO) queue that stores all the messages scheduled to be sent. Each sender periodically checks, following the exponential distribution with parameter $\frac{1}{\lambda_P}$, whether there is any scheduled message to be sent in their buffer. If there is a scheduled message, the sender pops this message from the buffer queue and sends it, otherwise a **drop** cover mes-

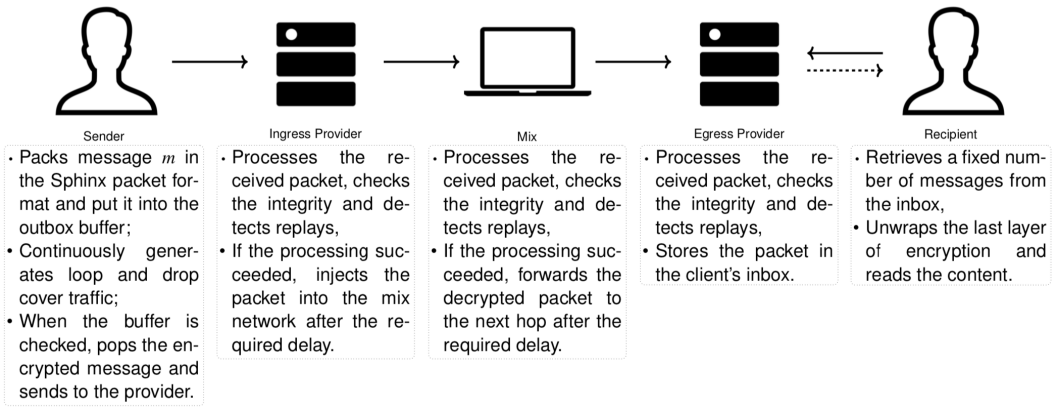


Figure 3.2: Sending a single message between two users using the Loopix system. The dotted line depicts retrieving of messages.

sage is generated, in the same manner as a regular message, and sent (depicted as the four middle blue, solid arrows in Figure 3.1). Drop cover messages are routed through the sender's provider and a random chain of mix nodes to a random destination provider. The destination provider detects the message is cover based on the special drop flag encapsulated into the packet header, and drops it. Thus, regardless of whether a user actually wants to send a message or not, there is always a stream of messages being sent according to a Poisson process $\mathsf{Pois}(\lambda_P)$.

• **Independent streams of cover traffic.** Moreover, independently from the above, all users emit separate streams of special indistinguishable types of cover messages, each of them following an independent Poisson process. The first type of cover messages are Poisson distributed **loops** emitted at rate λ_L. These are routed through the network and *looped back* to the senders (the upper four red arrows in Figure 3.1), by specifying the sending user as the recipient. These *"loops"* inspire the system's name. We denote the Poisson process responsible for loop cover messages as $\mathsf{Pois}(\lambda_L)$. Users also inject a separate stream of **drop** cover messages, defined as before, following the Poisson distribution $\mathsf{Pois}(\lambda_D)$.

Each mix also injects its own *loop* cover traffic, drawn from a Poisson process with rate $\mathsf{Pois}(\lambda_M)$, into the network. Mix servers inject mix packets that are looped through a path, made up of a subset of other mix servers and one randomly selected *provider*, back to the sending mix server, creating a second type of *"loop"*. This loop originates and ends in a mix server (shown as the lower four green arrows in Figure 3.1).

In Section 3.5 we examine how the *loops* and the *drop* cover messages help protect against passive and active attacks.

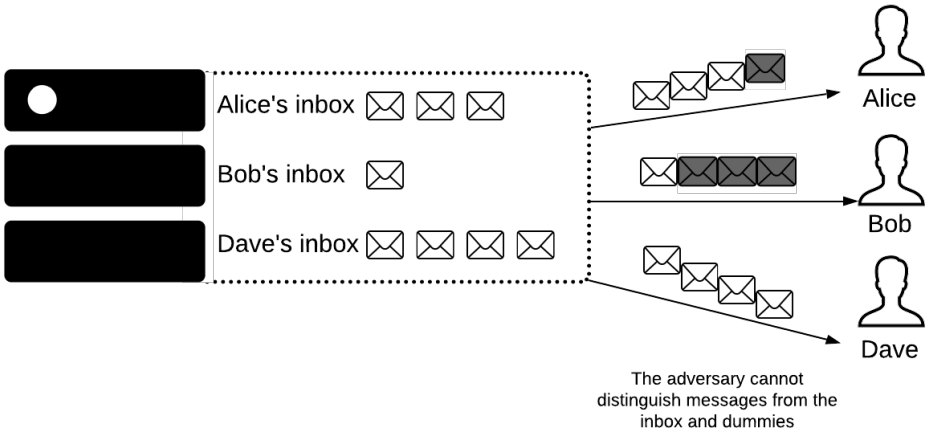


Figure 3.3: Message storage and retrival.

Processing messages. Upon receiving a packet, each node, i.e., each mix and provider, performs the operation of processing the Sphinx packet. While processing the packet, the server recomputes the shared secret and checks the MAC's correctness. If this integrity test fails, the packet is dropped. Otherwise, the unwrapping function returns the *replay detection tag* and the vector of *routing commands*, as well the *new packet*. The vector of *routing commands* includes, among others, the *routing flag*, the *address* of the next hop and the *delay*. After unwrapping the packet, the node checks whether the returned *replay detection tag* has been already seen and if so, drops the packet. This allows for detection and protection against *replay attacks*. Otherwise, the node saves the tag in a data structure that stores previously observed tags. Next, it checks whether the *routing flag* is set to Relay or Dest. The Dest flag means that the received message is a loop message transferred back to the node. In the case of the Relay flag, we consider two scenarios depending on whether the processing node is a mix or a provider. In the case of a mix, the decrypted *new packet* is send to the next hop, specified by *address*, after the *delay* has elapsed. In the case of a provider, the new packet is either forwarded as before or saved in the inbox of one of the provider's clients specified by the *address*.

Message storage and retrival. Providers do not forward the incoming mix packets to users but instead buffer them in clients' inboxes. Users, when online, *poll* providers at a fixed frequency or register their online status to download a fixed subset of stored messages, allowing for the reception of the off-line messages. Recall that cover loops are generated by users and traverse through the network and come back to the sender. Cover loops serve as a cover set of *outgoing* and *incoming*

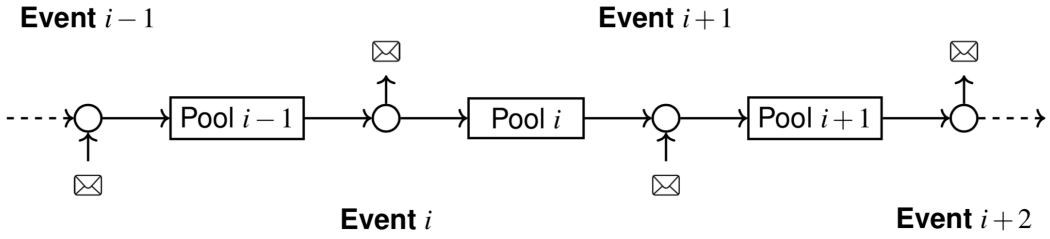


Figure 3.4: The Poisson Mix strategy mapped to a Pool mix strategy.

real messages. Whenever a user requests messages, their provider responds with a constant number of messages, which includes their cover loop messages and real messages. If the inbox of a particular user contains fewer messages than this constant number, the provider generates and sends *dummy* messages, indistinguishable from other type of messages, to the sender up to that number, see Figure 3.3.

3.4.3 The Poisson Mix Strategy

Loopix leverages cover traffic to resist traffic analysis while still achieving low- to mid-latency. To this end Loopix employs a mixing strategy that we call a *Poisson Mix*, to foil observers from learning about the correspondences between input and output messages. The Poisson Mix is a simplification of the Stop-and-go mix strategy [82]. A similar strategy has been used to model traffic in onion routing servers [179]. In contrast, recall that in Loopix each message is source routed through an independent route in the network.

The Poisson Mix functions as follows: mix servers listen for the incoming mix packets and received messages are checked for duplication and decoded using the mix node's private keys. The detected duplicates are dropped. Next, the mix node extracts a subsequent mix packet. Decoded mix packets are not forwarded immediately, but each of them is delayed according to a source pre-determined delay d_i. Honest clients chose these delays, independently for each hop, from an exponential distribution with a parameter μ that is assumed to be public and the same for all mix nodes. This parameter determines how long the message is queued in the mix. Thus, the end-to-end latency of the messages depends on the selected parameter μ.

Mathematical model of a Poisson Mix. Honest clients and mixes generate drop cover traffic, loop traffic, and messaging traffic following a Poisson process. Aggregating Poisson processes results in a Poisson process with the sum of their rates, therefore we may model the streams of traffic received by a Poisson mix as a

Poisson process. It is the superposition of traffic streams from multiple clients. It has a rate λ_n depending on the number of clients and the number of mix nodes.

Since this input process is a Poisson process and each message is independently delayed using an exponential distribution with parameter μ, the Poisson Mix may be modeled as an $M/M/\infty$ queuing system – for which we have a number of well known theorems [180]. We know that output stream of messages is also a Poisson process with the parameter λ_n as the the input process. We can also derive the distribution of the number of messages within a Poisson Mix in a *steady state* [57]. By the *steady state* we mean the state of the system in which all entities have already generated and processed messages for some reasonable period of time. By the convergence of the system to the equilibrium, this guarantees that the observed traffic closely follows the assumed distribution.

Lemma 1. *The mean number of messages in the Poisson Mix with input Poisson process* $\mathsf{Pois}(\lambda)$ *and exponential delay parameter* μ *at a steady state follows the Poisson distribution* $\mathsf{Pois}(\lambda/\mu)$.

These characteristics, which give the Poisson Mix its name, allow us to calculate the mean number of messages *perfectly* mixed together at any time, as well as the probability that the number of messages falls below or above certain thresholds.

The Poisson Mix, under the assumption that it approximates an $M/M/\infty$ queue is a stochastic variant of a pool mixing strategy [81]. Conceptually, every message sent or received leads to a pool within which messages are indistinguishable due to the memoryless property of the exponential delay distribution (described in Section 2.1). Intuitively, any two messages in the same pool are emitted next with equal probability – no matter how long they have been waiting. As illustrated in Figure 3.4, the receiving event $i-1$ leads to a pool of messages $i-1$, until the sending event i. From the perspective of the adversary observing all inputs and outputs, all messages in the pool $i-1$ are indistinguishable from each other. Only the presence of those messages in the pool is necessary to characterize the hidden state of the mix (not their delay so far). Relating the Poisson mix to a pool mix allows us to compute easily and exactly the entropy metric for the anonymity it provides [60] within a trace (used in Section 3.5.2). It also allows us to compute the likelihood that an emitted message was any specific input message used in our security evaluation.

Synchronous variant of Loopix. While Loopix operates asynchronously by design, we now also consider synchronous Loopix variant that operates in discrete rounds and thus cannot use the exponential mixing strategy, where delays attached to the packets are drawn from a continuous distribution. In a single round of the

synchronous system, the mixes gather packets, thus creating pools of packets. At the time of forwarding, the mix takes a number of packets from the pool and forwards them to the next destination. All the messages gathered in the pool during a single round are indistinguishable from each other, and each message has an equal probability of being sent. Hence, this mixing strategy leads to a geometric distribution [57], which similarly to exponential distribution is memoryless. Hence, the security analysis of mixing we present next can be applied both in the asynchronous and synchronous design. However, in contrast to the asynchronous Poisson mixes we presented, the synchronous variant of Loopix is vulnerable to the sleeper attacks [181].

3.5 Analysis of Loopix security properties

In this section we present the analytical and experimental evaluation of the security of Loopix and argue its resistance to traffic analysis and active attacks.

3.5.1 Passive attack resistance

Message Indistinguishability. Loopix relies on the Sphinx packet format [177] to provide bitwise unlinkability of incoming and outgoing messages from a mix server; it does not leak information about the number of hops a single message has traversed or the total path length; and it is resistant to tagging attacks.

For Loopix, we make minor modifications to Sphinx to allow auxiliary meta-information to be passed to different mix servers. Since all the auxiliary information is encapsulated into the header of the packet in the same manner as any meta-information was encapsulated in the Sphinx design, the security properties are unchanged. An external adversary and a corrupt intermediate mix node or a corrupt provider will not be able to distinguish *real* messages from *cover* messages of any type. Thus, the GPA observing the network cannot infer any information about the type of the transmitted messages, and intermediate nodes cannot distinguish real messages, drop cover messages or loops of clients and other nodes from each other. Providers are able to distinguish *drop* cover message destined for them from other messages, since they learn the *drop flag* attached in the most inner header of the packet. Each mix node learns the delay chosen by clients for this particular mix node, but all delays are chosen independently from each other.

Client-Provider unobservability. We now argue the *sender and receiver unobservability* against different adversaries in our threat model. Users emit payload

messages following a Poisson distribution with parameter λ_P. All messages scheduled for sending by the user are placed within a first-in-first-out buffer. According to a Poisson process, a single message is popped out of the buffer and sent, or a drop cover message is sent in case the buffer is empty. Thus, from an adversarial perspective, there is always traffic emitted modeled by $\mathsf{Pois}(\lambda_P)$. Since clients send also streams of cover traffic messages with rates λ_L for loops and λ_D for drop cover messages, the traffic sent by the client follows $\mathsf{Pois}(\lambda_P + \lambda_L + \lambda_D)$. Thus, we achieve perfect *sender unobservability*, since the adversary cannot tell whether a genuine message or a drop cover message is sent.

When clients query providers for received messages, the providers always send a constant number of messages to the client. If the number of messages in client's inbox is smaller than a constant threshold, the provider generates additional dummy messages. Thus, the adversary observing the client-provider connection cannot learn how many messages were in the user's inbox. Note that, as long as the providers are honest, the protection and *receiver unobservability* is perfect and the adversary cannot learn any information about the inbox and outbox of any client.

Corrupt providers. We distinguish the sender's and recipient's providers by calling them the ingress and egress providers respectively. If the *ingress provider* is compromised, all security properties of the Loopix system are still preserved, since the *ingress provider* observes a rate of traffic shaped by the Poisson distribution coming from the client and cannot distinguish whether the received packets carry real, loop or drop messages.

If the *egress provider* is malicious it can reveal to the adversary whether a particular client is receiving messages or not since the provider is responsible for managing the clients' inboxes. However, even an egress provider is still uncertain whether a received message is genuine or the result of a client loop – this cannot be determined from their bit pattern alone. Further statistical attacks may be possible, and we leave quantifying the exact information leakage against this threat model as future work. Thus, Loopix does not guarantee perfect *receiver unobservability* in the presence of a corrupted egress provider.

3.5.2 Poisson mix security

We continue our analysis of passive attacks by showing that a single honest Poisson mix provides a measure of *sender-receiver unlinkability*. From the properties of Poisson mix presented in Section 3.4.3, we know that the number of messages in the mix server at a steady state depends on the ratio of the incoming traffic (λ) and

the delay parameter (μ). The number of messages in each mix node at any time will on average be $\frac{\lambda}{\mu}$. However, an adversary observing the messages flowing into and out of a single mix node could estimate the exact number of messages within a mix with better accuracy – hindered only by the mix loop cover traffic.

3.5.2.1 Case 1: No mix loop cover traffic

We first consider, conservatively, the case where a mix node is not generating any loops and the adversary can count the exact number of messages in the mix. Let us define $o_{n,k,l}$ as an adversary A observing an empty mix in which n messages arrive and are mixed together. The adversary then observes an outgoing set of $n-k$ messages and can infer that there are now $k < n$ messages in the mix. Next, l additional messages arrive at the mix before any message leaves, and the pool now mixes $k+l$ messages, see Figure 3.5. The adversary then observes exactly one outgoing message m and tries to correlate it with any of the $n+l$ messages which she has observed arriving at the mix node.

The following lemma is based on the memoryless property of the Poisson mix. It provides an upper bound on the probability that the adversary A correctly links the outgoing message m with one of the previously observed arrivals in observation $o_{n,k,l}$.

Theorem 1. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Then*

$$\begin{aligned} Pr(m = m_1) &= \frac{k}{n(l+k)}, \\ Pr(m = m_2) &= \frac{1}{l+k}. \end{aligned} \tag{3.1}$$

Note that the last l messages that arrived at the mix node have equal probabilities of being the outgoing message m, independently of their arrival times. Thus, the arrival and departure times of the messages cannot be correlated, and the adversary learns no additional information by observing the timings. Note that $\frac{1}{l+k}$ is an upper bound on the probability that the adversary A correctly links the outgoing message to an incoming message. Thus, continuous observation of a Poisson mix leaks no additional information other than the number of messages present in the mix. We leverage those results for a single Poisson Mix to simulate the information propagated withing a the whole network observed by the adversary (c.f. Section 3.6).

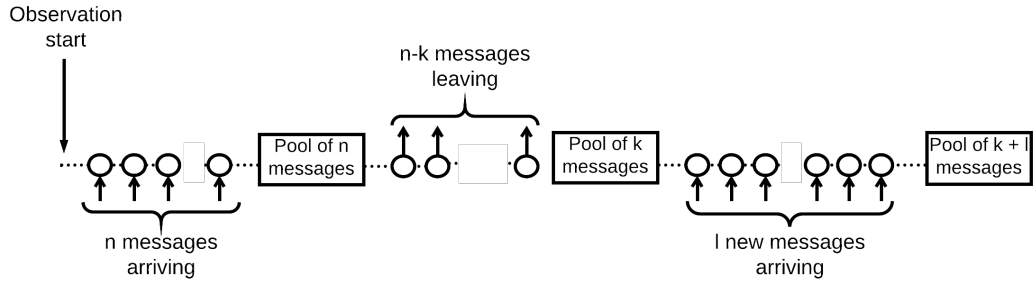


Figure 3.5: Observation $o_{n,k,l}$

Quantifying mix node anonymity using entropy metric. A common measure of anonymity is the *anonymity set*, which reflects the size of the set of other packets with which our message can be confused by the attacker. However, an adversary observing a mix for a while may assign different probabilities for each outgoing packet being linked to the observed incoming packet. Different probabilities of different members of the anonymity set reveal a lot of information to the attacker. Therefore, we quantify the anonymity of messages empirically, using the concept of Shannon entropy, an information theory based metric introduced by Danezis et al. [60, 61], which allows us to reason about the information contained in the probability distribution.

Definition of Shannon entropy. Let X be a discrete random variable over the finite set $\mathbb{X}$ with probability mass function $p(x) = \Pr(X = x)$. The Shannon entropy $H(X)$ of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in \mathbb{X}} p(x) \log p(x). \tag{3.2}$$

In a nutshell, entropy measures the unpredictability (randomness) of information content [54]. The information entropy $H(X)$ is the average amount of information conveyed by an event when considering an entire probability distribution of a discrete random variable. The more certain or deterministic the event is, the less information the variable contains. Hence, the increase in entropy is an increase in uncertainty. Entropy is zero when one outcome is certain to occur, and maximum when all event outcomes are equally probable.

In this section, we measure the entropy of a single mix node, while in Section 3.6 we study the end-to-end anonymity. In order to empirically measure the entropy, we record the traffic flow for a single mix node and compute the distribution of probabilities that the outgoing message is the adversary's target message. Given this distribution we compute the value of Shannon entropy, a measure of

unlinkability of incoming to outgoing messages.

Let $o_{n,k,l}$ be an observation as defined earlier for a mix pool at time t. We note that any outgoing message will have a distribution over being linked with past input messages, and the entropy H_t of this distribution is our anonymity metric. H_t can be computed incrementally given the size of the pool l (from previous mix rounds) and the entropy H_{t-1} of the messages in this previous pool, and the number of messages k received since a message was last sent:

$$H_t = H\left(\left\{\frac{k}{k+l}, \frac{l}{k+l}\right\}\right) + \frac{k}{k+l}\log k + \frac{l}{k+l}H_{t-1}, \tag{3.3}$$

for any $t > 0$ and $H_0 = 0$. Thus for sequential observations we can incrementally compute the entropy metric for each outgoing message, without remembering the full history of the arrivals and departures at the Poisson mix. We use this method to compute the entropy metric illustrated in Figure 3.6. For the computation we simulate the mix using the simpy package in Python. All data points are averaged over 50 simulations.

Simulation results. Figure 3.6 depicts the change of entropy against an increasing rate of incoming mix traffic λ. We simulate the dependency between entropy and traffic rate for different mix delay parameter μ (note, that $EX = \frac{1}{\mu}$) by recording the traffic flow and changing state of the mix node's pool. As expected, we observe that for a fixed delay, the entropy increases when the rate of traffic increases. The higher delay also results in an increase in entropy, denoting a larger potential anonymity set, since more messages are mixed together. The simulation shows how the rate of incoming traffic and assigned delay influence the entropy. Therefore, depending on the application we can adjust those parameters accordingly.

3.5.2.2 Case 2: Mix generates loop cover traffic

In case the mix node emits loop cover traffic, the adversary with observation $o_{n,k,l}$, tries to estimate the probability that the observed outgoing message is a particular *target* message she observed coming into the mix node. An outgoing message can be either input message or a loop message generated by the mix node – resulting in additional uncertainty for the adversary.

Theorem 2. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Let λ_M denote the rate*

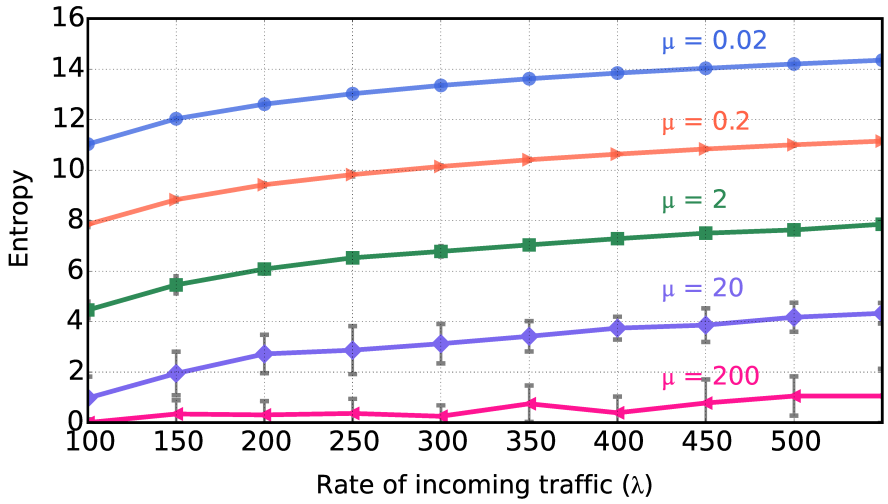


Figure 3.6: Entropy versus the changing rate of the incoming traffic for different delays with mean $\frac{1}{\mu}$. In order to measure the entropy we run a simulation of traffic arriving at a single Loopix mix node.

at which mix node generates loop cover traffic. Then,

$$\Pr(m = m_1) = \frac{k}{n} \cdot \frac{\mu}{(l+k)\mu + \lambda_M},$$
$$\Pr(m = m_2) = \frac{\mu}{(l+k)\mu + \lambda_M}.$$

Proof. Let us assume, that in mix node M_i there are n' messages at a given moment, among which is a target message m_t. Each message has a delay d_i drawn from the exponential distribution with parameter μ. The mix node generates loops with distribution $\text{Pois}(\lambda_M)$. The adversary observes an outgoing message m and wants to quantify whether this outgoing message is her target message. The adversary knows, that the output of the mix node can be either one of the messages inside the mix or its loop cover message. Thus, for any message m_t, the following holds

$$\Pr[m = m_t] = \Pr[m \neq loop] \cdot \Pr[m = m_t | m \neq loop] \tag{3.4}$$

We note that the next message m is a loop if and only if the next loop message is sent before any of the messages within the mix, i.e., if the sampled time for the next loop message is smaller than any of the remaining delays of all messages within the mix. We now leverage the memoryless property of the exponential distribution to model the remaining delays of all n' messages in the mix as fresh random samples

from the same exponential distribution.

$$\begin{aligned}
&\Pr[m \neq loop] = 1 - \Pr[m = loop] \\
&= 1 - \Pr[X < d_1 \wedge X < d_2 \wedge \ldots \wedge X < d_{n'}] \\
&= 1 - \Pr[X < min\{d_1, d_2, \ldots d_{n'}\}]
\end{aligned} \tag{3.5}$$

We know, that $d_i \sim Exp(\mu)$ for all $i \in \{1,\ldots,n'\}$ and $X \sim Exp(\lambda_M)$. Moreover, we know that the minimum of n independent exponential random variables with rate μ is an exponential random variable with parameter $\sum_i^{n'} \mu$. Since all the delays d_i are independent exponential variables with the same parameter, we have for $Y = \min\{d_1, d_2, \ldots d_{n'}\}$, $Y \sim Exp(n'\mu)$. Thus, we obtain the following continuation of Equation (3.5).

$$\begin{aligned}
\Pr[m \neq loop] = 1 - \Pr[X < Y] &= 1 - \int_0^\infty \Pr[X < Y | X = x] \Pr[X = x]\, dx \\
&= 1 - \int_0^\infty \Pr[x < Y]\, \lambda_M e^{-\lambda_M x} dx \\
&= 1 - \int_0^\infty e^{-n'\mu x} \lambda_M e^{-\lambda_M x} dx
\end{aligned} \tag{3.6}$$

$$= 1 - \frac{\lambda_M}{\lambda_M + n\mu} = \frac{n'\mu}{n'\mu + \lambda_M}$$

Since the probability to send a loop depends only on the *number* of messages in a mix, but not on which messages are in the mix, this probability is independent of the probability from Theorem 1. Theorem 2 follows directly by combining Theorem 1 and Equation (3.7), with $n' = k + l$. We get for messages m_1 that previously were in the mix,

$$\begin{aligned}
\Pr[m = m_1] &= \Pr[m \neq loop] \cdot \Pr[m = m_1 | m \neq loop] \\
&= \frac{(k+l)\mu}{(k+l)\mu + \lambda_M} \cdot \frac{k}{n(k+l)} = \frac{k}{n} \cdot \frac{\mu}{(k+l)\mu + \lambda_M}.
\end{aligned} \tag{3.7}$$

Analogously, we get for m_2,

$$\begin{aligned}
\Pr[m = m_2] &= \Pr[m \neq loop] \cdot \Pr[m = m_2 | m \neq loop] \\
&= \frac{(k+l)\mu}{(k+l)\mu + \lambda_M} \cdot \frac{1}{k+l} = \frac{\mu}{(k+l)\mu + \lambda_M}.
\end{aligned} \tag{3.8}$$

This concludes the proof. □

We conclude that the loops generated by the mix node obfuscate the adver-

sary's view and decrease the probability of successfully linking input and output of the mix node. In Section 3.5.3, we next show that those types of loops also protect against active attacks.

3.5.3 Active attack resistance

Lemma 1 in Section 3.4.3 gives the direct relationship between the expected number of messages in a mix node, the rate of incoming traffic, and the delay induced on a message while transiting through a mix. By increasing the rate of cover traffic, λ_D and λ_L, users can collectively maintain strong anonymity with low message delay. However, once the volume of real communication traffic λ_P increases, users can tune down the rate of cover traffic in comparison to the real traffic, while maintaining a small delay and be confident their messages are mixed with a sufficient number of messages.

In the previous section, we analyze the security properties of Loopix when the adversary observes the state of a single mix node and the traffic flowing through it. We show, that the adversary's advantage is bounded due to the indistinguishability of messages and the memoryless property of the Poisson mixing strategy. We now consider an attack at a mix node where an adversary blocks all but a target message from entering in order to follow the target message when it exits the mix node. This is referred to as an *(n-1) attack* [68].

A mix node needs to distinguish between an active attack and loop messages dropped due to congestion. We assume that each mix node chooses some public parameter r, which is a fraction of the number of loops that are expected to return. If the mix node does not see this fraction of loops returning they alter their behaviour. In extremis such a mix could refuse to emit any messages – but this would escalate this attack to full denial-of-service. A gentler approach involves generating more cover traffic on outgoing links [182].

To attempt an *(n-1) attack*, the adversary could simply block all incoming messages to the mix node except for a target message. The Loopix mix node can notice that the self-loops are not returning and deduce it is under attack. Therefore, an adversary that wants to perform a stealthy attack has to be judicious when blocking messages, to ensure that a fraction r of loops return to the mix node, i.e. the adversary must distinguish loop cover traffic from other types of traffic. However, traffic generated by mix loops is indistinguishable from other network traffic and they cannot do this better than by chance. Therefore given a threshold $r = \frac{\lambda_M}{s}, s \in \mathbb{R}_{>1}$ of expected returning loops when a mix observes fewer returning it deploys appropriate countermeasures.

We analyze this strategy: since the adversary cannot distinguish loops from other traffic the adversary can do no better than block traffic uniformly such that a fraction $R = \frac{\lambda}{s} = \frac{\lambda_R + \lambda_M}{s}$ enter the mix, where λ_R is the rate of incoming traffic that is not the mix node's loops. If we assume a steady state, the target message can expect to be mixed with $\frac{\lambda_R}{s \cdot \mu}$ messages that entered this mix, and $\frac{\lambda_M}{\mu}$ loop messages generated at the mix node. Thus, the probability of correctly blocking a sufficient number of messages entering the mix node so as not to alter the behaviour of the mix is:

$$\Pr(x = \text{target}) = \frac{1}{\lambda_R / s \cdot \mu + \lambda_M / \mu} = \frac{s\mu}{s\lambda_M + \lambda_R}$$

Due to the stratified topology, providers are able to distinguish mix loop messages sent from other traffic, since they are unique in not being routed to or from a client. This is not a substantial attack vector since mix loop messages are evenly distributed among all providers, of which a small fraction are corrupt and providers do not learn which mix node sent the loop to target it.

3.6 End-to-End Anonymity Evaluation

So far we presented how a single mix node in the Loopix system ensures unlinkability of incoming and outgoing messages by measuring the entropy of information inferred by the observing adversary, or how the *loop* packets allow the mix nodes, as well the clients, detect active $(n-1)$ attacks. Analysing a single mix node is equivalent to analysing a mix network as a black box. However, a global adversary can observe the entire network traffic and the intermediate links between nodes, and hence see all incoming and outgoing traffic in every layer, or when users send and receive messages. Therefore, in this section, we focus on the analysis of end-to-end anonymity offered by Loopix.

In order to do that, we build a mix network discrete simulator, which given different configuration parameters of the network topology, number of users or latency-overhead demands, allows to empirically evaluate the anonymity and performance properties of Loopix. Our simulator realizes the Loopix system design presented in Section 3.4.[2] It is implemented in Python3, and the interaction between the different network components (e.g., clients, mixes) is simulated using the discrete-event Simpy framework. In general, our simulator allows analysing how anonymity, latency, and bandwidth overhead are affected by various scenarios of

[2]The code of the simulator is currently in a private Git repository but can be shared with readers, who should contact the author if interested.

deployment of such networks, like volume of traffic or throughput, latency requirements, packet size or network topology.

The information-theoretic anonymity metric presented in section Section 3.5.2 can be also used to measure the end-to-end anonymity offered by Loopix. However, the limitation of this metric is that it allows computing anonymity per single packet in one run of the system, hence it does not give us any insight about the information leaked by long-term patterns occurring during repeated uses of the system. Therefore, we introduce a novel approach for measuring the end-to-end anonymity of anonymous communication systems, the *sender-receiver third-party unlinkability* metric, which allows easy analysis of cumulative privacy loss over multiple rounds.

3.6.1 Sender-Receiver Third-party Unlinkability

We evaluate the *sender-receiver third-party unlinkability* of the full Loopix system through an empirical analysis of the propagation of messages in the network. Our key metric is the *expected difference in likelihood* that a message leaving the last mix node is sent from one sender in comparison to another sender. We use this metric to measure the chances of the adversary correctly correlating the communicating users if the adversary is given more information in advance, meaning - the adversary already knows that either Alice or Bob are communicating with Eva. This is a large (but realistic) advantage for the adversary, and we want to investigate how much such information impacts the anonymity offered by the network.

To quantify the *sender-receiver third-party unlinkability* we consider the following security experiment. The adversary selects two target senders, S_0 and S_1, and a target recipient R. One of those senders, selected secretly by the challenger, will send messages to the target recipient. The adversary observing the network traffic tries to infer, who of those two senders communicates with the target recipient. Given two probabilities $p_0 = \Pr[S_0]$ and $p_1 = \Pr[S_1]$ that the message was sent by senders S_0 and S_1, respectively, we calculate

$$\varepsilon = |\log(p_0/p_1)|. \tag{3.9}$$

Intuitively, we can think about ε as the maximum leakage the adversary can learn from observing the system and both the events S_0 and S_1 (so how much the events differ). As δ we define the probability by which this leakage exceeds ε (small ε and δ values are better).

Experimental Evaluation. To approximate the probabilities p_0 and p_1, we proceed as follows. We simulate $U = 100$ senders that generate and send messages (both payload and cover messages) with a rate $\lambda = 2$. Among them are two challenge senders S_0 and S_1 that send payload messages at a constant rate, i.e, they add one messages to their sending buffer every time unit.

Whenever a challenge sender S_0 or S_1 sends a payload message from its buffer, we tag the message with a label S_0 or S_1, respectively. All other messages, including messages from the remaining 98 clients and the cover messages of S_0 and S_1 are unlabeled. At every mix we track the probability that an outgoing message is labeled S_0 or S_1, depending on the messages that entered the mix node and the number of messages that already left the mix node, as in Theorem 1. Thus, messages leaving a mix node carry a probability distribution over labels S_0, S_1, or 'unlabeled'. Corrupt mix nodes, assign to outgoing messages their input distributions. The probabilities naturally add up to 1. For example, a message leaving a mix can be labeled as $\{S_0 : 12\%, S_1 : 15\%, \text{unlabeled} : 73\%\}$.

In a burn-in phase of 2500 time units, the 98 senders without S_0 or S_1 communicate. Then we start the two challenge senders and then simulate the network for another 100 time units, before we compute the *expected difference in likelihood* metric. We pick a final mix node and using probabilities of labels S_0 and S_1 for any message in the pool we calculate ε as in Equation (3.9).

This is a conservative approximation: we tell the adversary which of the messages leaving senders S_0 and S_1 are payload messages; and we do not consider mix or client loop messages confusing them[3]. However, when we calculate our anonymity metric at a mix node we assume this mix node to be honest.

Results. We compare our metric for different parameters: depending on the delay parameter μ, the number of layers in our topology l and the percentage of corrupt mix nodes in the network. All simulations are averaged over 100 repetitions and the error bars are the standard deviation.

Delay. Increasing the average delay (by decreasing parameter μ) with respect to the rate of message sending λ immediately increases anonymity (decreases ε) (Figure 3.7). For $\mu = 2.0$ and $\lambda/\mu = 1$, Loopix still provides a weak form of anonymity. As this fraction increases, the log likelihood ratio grow closer and closer to zero. We consider values $\lambda/\mu \geq 2$ to be a good choice in terms of anonymity.

Number of layers. By increasing the number of layers of mix nodes, we can fur-

[3]The soundness of our simplification can be seen by the fact that we could tell the adversary which messages are loops and the adversary could thus ignore them. This is equivalent to removing them, as an adversary could also simulate loop messages.

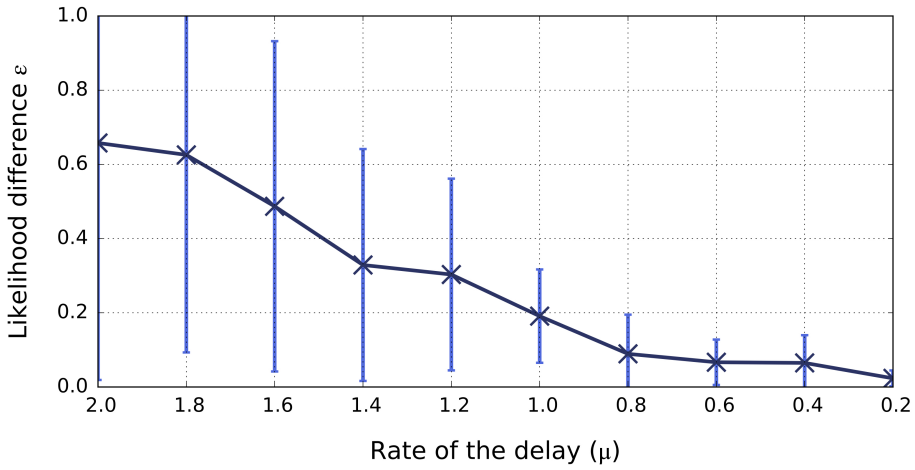


Figure 3.7: Likelihood difference ε vs the delay parameter μ of mix nodes. We use $\lambda = 2$, a topology of 3 layers with 3 nodes per layer and no corruption.

ther strengthen the anonymity of Loopix users. As expected, using only one or two layers of mix nodes leads to high values of adversary advantage ε. For a increasing number of layers, ε approaches zero (Figure 3.8). We consider a number of 3 or more layers to be a good choice. We believe the bump between 5–8 layers is due to messages not reaching latter layers within 100 time units. Results from experiments with increased duration do not display such a bump.

Corruption. Finally, we analyze the impact that corrupt mix nodes have on the adversary advantage ε (Figure 3.9). We assume that the adversary randomly corrupts mix nodes. Naturally, the advantage ε increases with the percentage of corrupt mix nodes in the network. In a real-world deployment we do not expect a large fraction of mix nodes to be corrupt. While the adversary may be able to observe the entire network, to control a large number of nodes would be more costly.

3.7 Performance Evaluation

Implementation. We implement the Loopix system prototype in 4000 lines of Python2.7 code for *mix nodes*, *providers* and *clients*, including unit-tests, deployment, and orchestration code. Loopix source code is available under an open-source license.[4] We use the Twisted15.5.0 network library for networking; as well as the Sphinx mix packet format and the cryptographic tools from the *petlib*library.[5,6] We

[4] https://github.com/UCL-InfoSec/loopix
[5] http://sphinxmix.readthedocs.io/en/latest/
[6] http://petlib.readthedocs.org

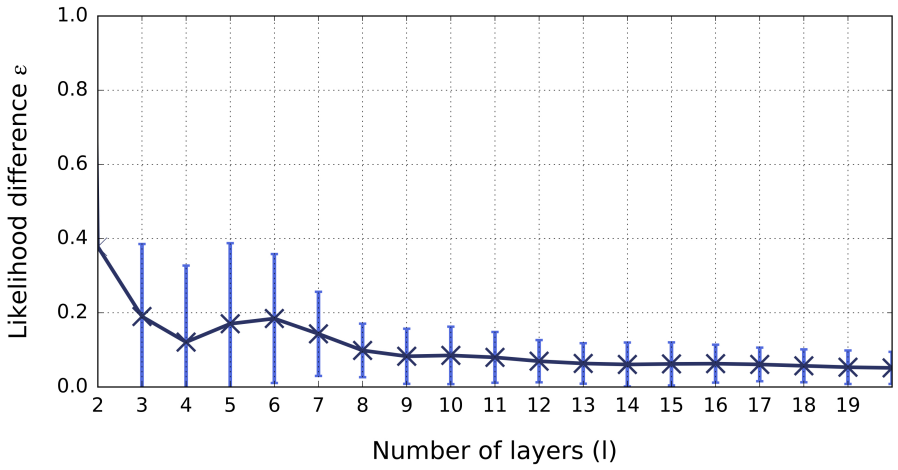


Figure 3.8: Likelihood difference ε vs the number of layers of mix nodes with 3 mix nodes per layer. We use $\lambda = 2$, $\mu = 1$, and no corruption.

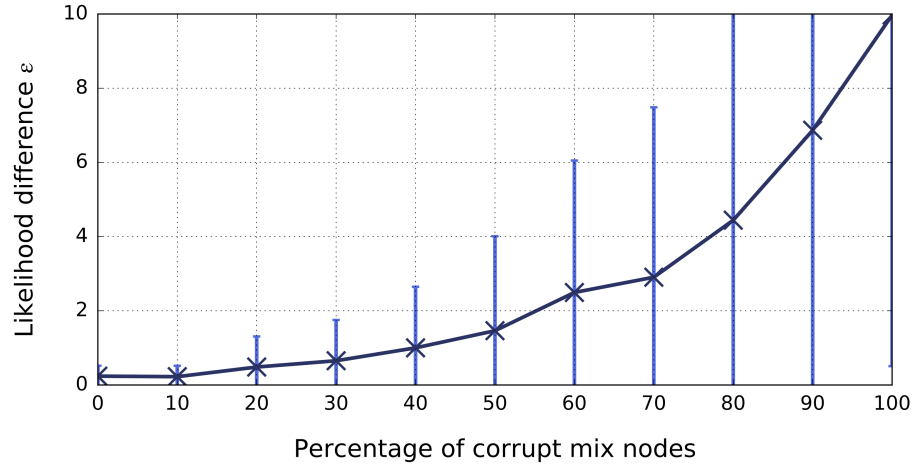


Figure 3.9: Likelihood difference ε vs the percentage of (passively) corrupted mix nodes. We use $\lambda = 2$, $\mu = 1$ and a topology of 3 layers with 3 nodes per layer.

modify Sphinx to use NIST/SEGS-p224 curves and to accommodate additional information inside the packet, including the delay for each hop and auxiliary flags. We also optimize the Sphinx implementation leading to processing times per packet of less than $1ms$.

The most computationally expensive part of Loopix is messages processing and packaging, which involves cryptographic operations. Thus, we implement Loopix as a multi-thread system, with cryptographic processing happening in a thread pool separated from the rest of the operations in the main thread loop. To recover from congestion we implement active queue management based on a PID

controller and we drop messages when the size of the queue reaches a (high) threshold.

Experimental Setup. We present an experimental performance evaluation of the Loopix system running on the AWS EC2 platform. All mix nodes and providers run as separate instances. Mix nodes are deployed on m4.4xlarge instances running EC2Linux on 2.3 GHz machines with 64 GB RAM memory. Providers, since they handle more traffic, storage and operations, are deployed on m4.16xlarge instances with 256 GB RAM. We select large instances to ensure that the providers are not the bottleneck of the bandwidth transfer, even when users send messages at a high rate. This reflects real-world deployments where providers are expected to be well-resourced. We also run one m4.16xlarge instance supporting 500 clients. We only show results for 500 clients, due to limitations of our experimental hardware setup such as ports and memory. A real world deployment of Loopix would scale to a larger client base. We believe that our empirical analysis is a more accurate assessment of real-world performance than those reported by other works, e.g. [133, 135], which depend on simplish extrapolation. In order to measure the system performance, we run six mix nodes, arranged in a stratified topology with three layers, each layer composed of two mix nodes. Additionally, we run four providers, each serving approximately 125 clients. The delays of all the messages are drawn from an exponential distribution with parameter μ, which is the same for all mix servers in the network. All measurements are taken from network traffic dumps using tcpdump.

Bandwidth. First, we evaluate the increase of bandwidth of mix nodes by measuring the rate at which a single mix node processes messages, for an increasing overall rate at which users send messages. We set up the fixed delay parameter $\mu = 1000$ (s.t. the average delay is $1\,ms$). We have 500 clients actively sending messages at rate λ each, which is the sum of payload, loop and drop rates, i.e., $\mathsf{Pois}(\lambda) = \mathsf{Pois}(\lambda_L + \lambda_D + \lambda_P)$. We start our simulation with parameters $\lambda_L = \lambda_D = 1$ and $\lambda_P = 3$ messages per minute for a single client. Mix nodes send loop cover traffic at rate starting from $\lambda_M = 1$. Next, we periodically increase each Poisson rate by another 2 messages per minute. Each packet sent through the network has a size of a few kilobytes only, but this size is a parameter that can, of course, be increased to fit the needs of a particular application. In order to measure the overall bandwidth, i.e. the number of all messages processed by a single mix node, we use the network packet analyzer tcpdump. Since real and cover message packets are indistinguishable, we measure the good throughput by encapsulating an additional, temporary,

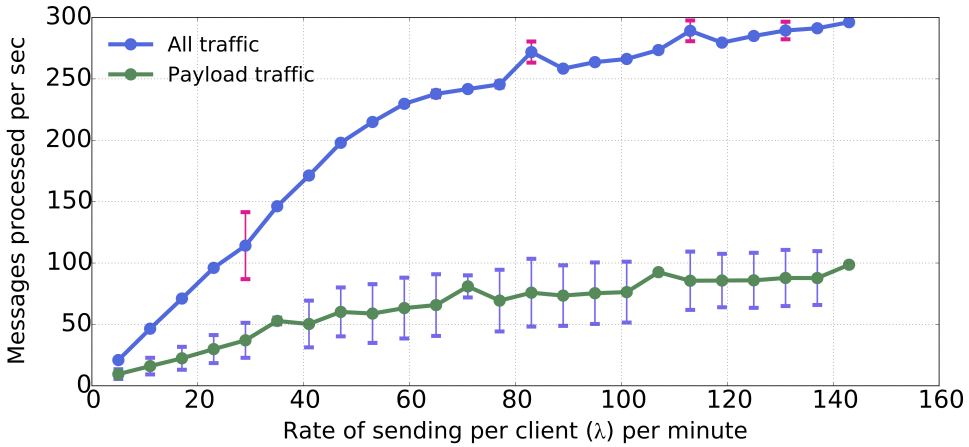


Figure 3.10: Overall bandwidth and good throughput per second for a single mix node.

typeFlag in the packet header for this evaluation, which leaks to the mix the message type—real or cover—and is recorded. Knowing the parameters λ_P, λ_L, and λ_D the adversary can try to estimate how many messages on average in the outgoing stream are real, loop or drop messages. However, the average estimation does not give the adversary any significant information, since the outgoing traffic may contain various numbers of each type of message which an adversary is not able to distinguish between.

Figure 3.10 illustrates the number of total messages and the number of payload messages that are processed by a single mix node versus the overall sending rate λ of a single user. We observe that the bandwidth of the mix node increases linearly until it reaches around 225 messages per second. After that point the performance of the mix node stabilizes and we observe a much smaller growth. We highlight that the amount of *real* traffic in the network depends on the parameter λ_P within λ. A client may chose to tune up the rate of real messages sent, by tuning down the rate of loops and drop messages – at the potential loss of security in case less cover traffic is present in the system overall. Thus, depending on the size of the honest user population in Loopix, we can increase the rate of goodput.

Latency Overhead & Scalability End-to-end latency overhead is the cost of routing and decoding relayed messages, without any additional artificial delays. We run simulations to measure its sensitivity to the number of users participating in the system. We measure the time needed to process a single packet by a mix node, which is approximately $0.6\,ms$. This cost is dominated by the scalar multiplication of an elliptic curve point and symmetric cryptographic operations. For the end-to-

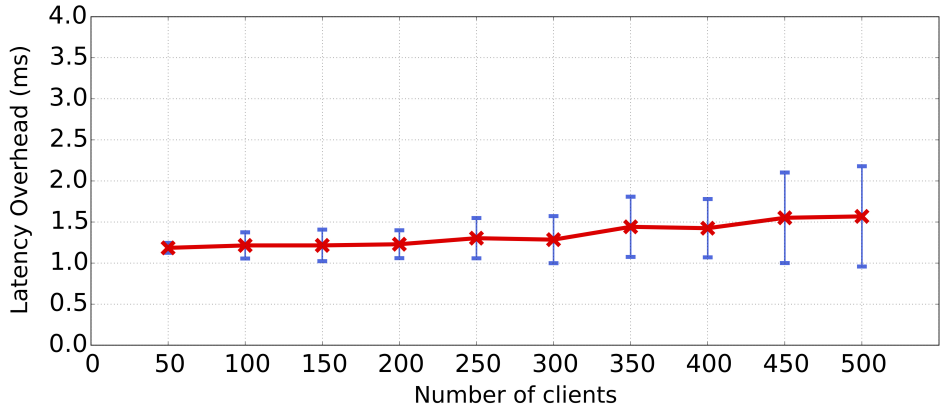


Figure 3.11: Latency overhead of the system for rates $\lambda_P = \lambda_L = \lambda_D = 10, \lambda_M = 10$ per minute, and no additional delay added to the messages by the senders.

end measurement, we run Loopix with a setup where all users have the same rates of sending real and cover messages, such that $\lambda_P = \lambda_D = \lambda_L = 10$ messages per minute and mix servers generate loops at rate $\lambda_M = 10$ messages per minute. All clients set a delay of 0.0 seconds for all the hops of their messages – to ensure we only measure the system overhead, not the artificial mixing delay.

Figure 3.11 shows that increasing the number of online clients, from 50 to 500, raises the latency overhead by only $0.37\,ms$. The variance of the processing delay increases with the amount of traffic in the network, but more clients do not significantly influence the average latency overhead. Neither the computational power of clients nor mix servers nor the communication between them seem to become bottlenecks for these rates. Those results show that the increasing number of users in the network does not lead to any bottleneck for our parameters. The measurements presented here are for a network of 6 mix nodes, however we can increase the system capacity by adding more servers. Thus, Loopix scales well for an increasing number of users.

We also investigate how increasing the delays through Poisson Mixing with $\mu = 2$ affects the end-to-end latency of messages given setup where $\lambda_P = \lambda_L = \lambda_D =$ 60 per minute and $\lambda_M = 60$ per minute.. We measure this latency through timing mix heartbeat messages traversing the system. Figure 3.12 illustrates that when the mean delay $1/\mu$ sec. is higher than the processing time ($\sim 1\,ms - 2\,ms$), the end-to-end latency is determined by this delay, and follows the Gamma distribution with parameter being the sum of the exponential distribution parameter over the number of servers on the path. The good fit to a gamma distribution provides evidence that the implementation of Loopix is faithful to the queuing theory models our analysis assumes.

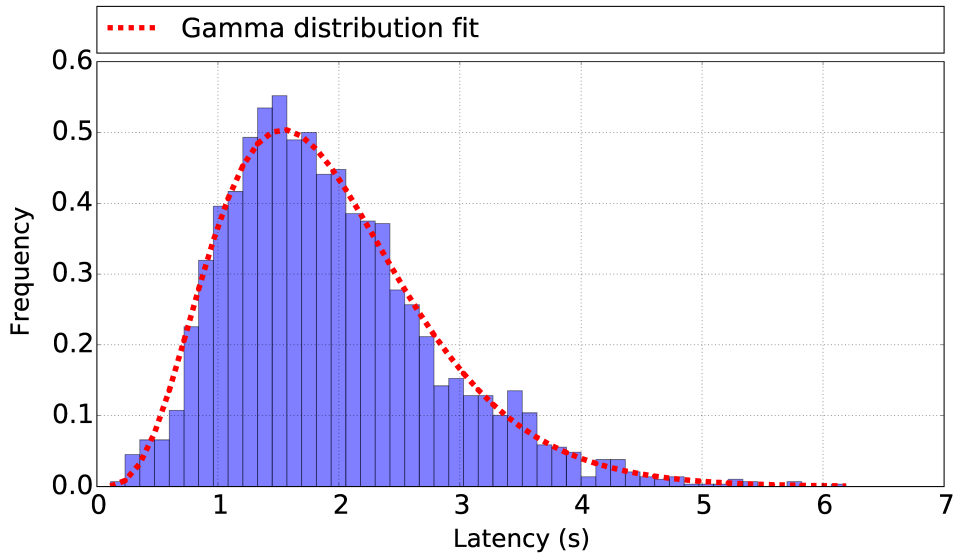


Figure 3.12: End-to-end latency histogram measured through timing mix node loops. The latency of the message is determined by the assigned delay and fits the Gamma distribution with mean 1.93 and standard deviation 0.87.

3.8 Comparison with Related Work

In Section 2.3, we presented a detailed survey of the literature on decentralized anonymous communication systems. In this section, we outline the main differences, both in terms of the security properties and performance capabilities, between Loopix and the designs described in Section 2.3. A summary is provided in Table 3.2.

	Low Latency	Low Communication Overhead	Scalable Deployment	Asynchronous Messaging†	Active Attack Resistant	Offline Storage*	Resistance to GPA
Loopix	✓	✓	✓	✓	✓	✓	✓
(1) Dissent [145]	✗	✗	✗	✗	✓	✗	✓
(2) Vuvuzela [133]	✗	✗	✓	✗	✓	✗	✓
(3) Riposte [146]	✗	✗	✓	✗	✓	✗	✓
(4) Atom [150]	✗	✓	✓	✗	✓	✗	✓
(5) Riffle [147]	✓	✓	✗	✗	✓	✗	✓
(6) AnonPoP [137]	✗	✓	✓	✗	✗	✓	✓
(7) Pung [138]	✗	✗	✗	✗	✓	✓	✓
(8) Tor [43]	✓	✓	✓	✓	✗	✗	✗
(9) Aqua [152]	✓	✓	✓	✓	✗	✗	✗
(10) Stadium [135]	✗	✓	✓	✗	✓	✗	✓
(11) Karaoke [136]	✗	✓	✓	✗	✓	✗	✓
(12) XRD [144]	✗	✗	✗	✗	✓	✗	✓

Table 3.2: Comparison of Loopix and modern anonymous communication systems.[7]

We start our comparison with **Tor** [43], as it is the most popular low-latency anonymity system, with almost 2 million users daily. While Tor offers low-latency

and scalability, its security properties are limited to the local adversaries. As stated in the Tor paper *'Tor does not claim to completely solve end-to-end timing or intersection attacks'*, hence Tor can be easily defeated by a network adversary who can monitor both the entry and exit points of the network and perform end-to-end correlation of traffic flows, or has a global view on the network [44, 45, 46, 47, 48]. Its privacy guarantees weaken further if the adversary has capabilities to launch active attacks. In comparison, Loopix resists both traffic analysis and active attacks, yet similarly to Tor scales horizontally, and can be also used for low-latency communication.

Likewise Loopix, **Vuvuzela** [133] protects against both passive adversaries and active attacks on communication links as long as there is one honest mix node. Since Vuvuzela operates in rounds, offline users lose the ability to receive messages. Loopix does not operate in rounds, thus the end-to-end latency can be significantly smaller than in Vuvuzela, depending on the delay parameter the senders choose. Moreover, Loopix allows off-line users to receive messages and uses parallel mix nodes to improve the scalability of the network. In Vuvuzela on the other hand, all messages must traverse a single chain of relay servers, however, this comes at a non-trivial bandwidth cost, both for the servers and network users. Moreover, Vuvuzela does not aim to prevent servers from tampering with the users' messages, and as a result, a malicious server can drop all but one honest user's messages (hence perform (n-1) attack). Such attack cannot occur in Loopix thanks to the *loop cover traffic*.

Stadium [135] and **AnonPop** [137] refine Vuvuzela; both operating in rounds making the routing of messages dependent on the dynamics of others. To increase its scalability in Stadium mixing is parallelized, i.e., each mix receives only a fraction of the overall traffic, mixes those messages, and then again splits the messages among other servers - this is a similar approach to the use of Stratified topology proposed earlier in Loopix. Stadium lacks also offline storage, while AnonPop does not protect against active attacks. Loopix provides both properties, and because it operates continuously it avoids user synchronization issues.

Cmix [142] resists traffic analysis and intersection attacks, similarly to Loopix. However, all traffic is relayed via a fixed-cascade, hence Cmix does not scale horizontally. Moreover, to allow for low-latency communication, it requires long precomputation phase. Furthermore, Cmix leaks information about how many mes-

[7] By *, we mean if the design intentionally incorporates provisions for delivery of messages when a user is offline, perhaps for a long period of time. By †, we mean that the system operates continuously and does not depend on synchronized rounds for its security properties and users do not need to coordinate to communicate together.

sages each user received and is vulnerable to tagging attacks and insider attacks.

XRD [144] is an anonymous messaging system, that scales horizontally, by distributing the workload across many parallel mix chains, similarly to Loopix. XRD offers strong security properties, by ensuring that every pair of users intersects at one of the chains. However, to guarantee that each user has to send $O(\sqrt{N})$ messages given the network of N mix servers. This imposes a high overhead for the users and increases significantly the workload of each mix server. Therefore, the cost of adding a single user to the system is expensive, which is a large limitation for deployment. Hence, XRD has, two main drawbacks compared to Loopix; it does not scale well with the increasing user base and introduces significant latency overheads. At the time of writing this work, there is also no implementation of XRD, which would allow comparing its overall performance.

Although **Riposte** [146], which is based on a *write* PIR scheme in which users write their messages anonymously into a database, enjoys low communication-overhead and protects against traffic analysis and denial of service attacks, it requires long epochs and a small number of clients writing into the database simultaneously. In contrast to Loopix, it is suitable only for high-latency applications. Additionally, Riposte imposes a large overhead on the clients, who must send a message proportional to the square root of all the collection of all clients' data, and requires days to build a strong anonymity set.

Dissent [91], based on DC-networks [91], offers resilience against a GPA and some active attacks, but at significantly higher delays than Loopix and scales to only several thousand clients, while our system can scale to millions of users.

Riffle [147] proposes a new verifiable shuffle technique to achieve sender anonymity. Using PIR, Riffle guarantees receiver anonymity in the presence of an active adversary, as well as both sender and receiver anonymity, but it cannot support a large user base. Riffle also utilizes rounds to protect against traffic analysis attacks, which introduce synchronization issues. Moreover, unlike Loopix, Riffle is not designed for Internet-scale anonymous communication, but for only supporting intra-group communication.

Atom [150] combines a number of novel techniques to provide mid-latency communication, strong protection against passive adversaries and uses zero-knowledge proofs between servers to resist active attacks. Unlike Loopix, Atom is designed for latency tolerant uni-directional anonymous communication applications with only sender anonymity in mind. Besides being resistant to traffic analysis, Atom defends also against active attacks performed by malicious servers. However, Atom does not protect against more sophisticated attack, like for example intersection attacks [59], which Loopix does. In terms of latency Loopix also

outperforms Atom, which due to expensive cryptographic primitives and routing the messages through hundreds of servers in series, introduces latency overheads in the order of tens of minutes for a few million users. In contrast, Loopix's latency overhead decreases when the number of users increases, since the exponential delay per mix can be tunned down up to seconds or even milliseconds.

Pung [138] hides all meta-data associated with user's conversations, even against adversaries who are capable to control all of the communication infrastructures, hence offers a stronger privacy notion that our Loopix. However, this is achieved at the cost of performance, which grows superlinearly with the number of users, and limited throughput. This results in high latency.

Aqua [152], on the other hand, aims to provide low-latency and traffic analysis resistance for peer-to-peer file sharing applications, however, it offers only $k-anonymity$ security. Therefore, it suffers from similar security problems to Tor, in which adversaries controlling both ends of the circuit can easily deanonymize clients.

While the literature on anonymous communication offers a wide range of system designs, with different security or performance properties, as we saw in this section, Loopix is the first mix network design which ensures strong privacy properties of online communication, yet at the same time scales to millions of users, and allows to low-latency communication. In Loopix, the increasing number of system users does not only contribute to better privacy, as the anonymity set grows, but also makes the system faster, as less additional delays need to be added to anonymize the traffic, and the volume of cover traffic can be tunned down. This is a property which was not offered by any of the previous designs.

3.9 Discussion

Deployment of Loopix system. The Loopix design became the core mix network design for the deployment of anonymous communication infrastracture developed by the PANORAMIX project [183]. Loopix achieves its stated security and performance goals. However, there are many other facets of the design space that have been left for future work, for instance, reliable message delivery, session management. Therefore, Loopix was further enhanced and deployed in the PANORAMIX Katzenpost mix network, including the (1) mechanism for reliable message delivery using Single-use Reply Blocks (SURB) [177] to send acknowledgements, (2) integration with email client, (3) message fragmentation and retransmission protocol, (4) public key infrastructure etc. The specification of Katzenpost

mix network can be found in the following open-source repositories:

- Katzenpost Mix Network End-to-end Protocol Specification [184]
- Katzenpost Mix Network Specification [185]
- Katzenpost Mix Network Public Key Infrastructure Specification [186]
- Sphinx Mix Network Cryptographic Packet Format Specification [187]

Moreover, the Loopix design is also adopted by the Nym Technologies [188] company, which is building an open-source, permissionless, and incentivized infrastructure for a privacy-enhanced internet. At the time of writing this thesis, the Nym's testnet around 900 mix nodes running.

Reply messages Loopix currently allows two methods if the receiver does not already know the sender a priori: we either attach the address of the sender to each message payload, or provide a single-use anonymous reply block [74, 177], which enables different use-cases. We leave the analysis of replies to messages as future work.

Secure lookup system It is also apparent that an efficient and secure private lookup system, one that can deliver network state and keying information to its users, is necessary to support modern anonymous communications. Proposals of stand-alone 'presence' systems such as DP5 [189] and MP3 [190] provide efficient lookup methods, however, we anticipate that tight integration between the lookup and anonymity systems may bring mutual performance and security benefits, which is another avenue for future work.

Anonymity trillema in practice [50] As shown in Section 3.5.1, the security of Loopix heavily depends on the ratio of the rate of traffic sent through the network and the mean delay at every mix node. Optimization of this ratio is application dependent. For applications with small number of messages and delay tolerance, a small amount of cover traffic can guarantee security. Therefore, it is important to understand how the volume of traffic and network parameters influence the anonymity and performance of the communication.

3.10 Conclusion

We have described Loopix, a mix network-based communication system which explores the design space frontiers of low-latency mixing. We presented how Loopix balances cover traffic and message delays to achieve a tunable tradeoff between real

traffic and cover traffic, and between latency and good anonymity. Low-latency incentivizes early adopters to use the system, as they benefit from good performance. Moreover, the cover traffic introduced by both clients and mix servers provides security in the presence of a smaller user-base size. In turn, this promotes growth in the user-base leading on one hand to greater security [191], and on the other a tuning down of cover traffic over time.

Loopix is the first system to combine a number of best-of-breed techniques: we provide definitions inspired by AnoA [174] for our security properties; improve the analysis of simplified variants of stop-and-go-mixing as a Poisson mix [82]; use restricted topologies to promote good mixing [164] and deploy modern active attack mitigations based on loops [182]. We also use modified modern cryptographic packet formats, such as Sphinx [177], for low information leakage. The result of composing these different techniques – previously explored as separate and abstract design options – is a design that is strong against global network level adversaries without the very high-latencies traditionally associated with mix systems [73, 74]. Thus, Loopix revitalizes message-based mix systems and makes them competitive once more against onion routing [192] based solutions that have dominated the field of anonymity research since Tor [43] was proposed in 2004.

Chapter 4

Detecting malicious mix nodes

In this chapter, we introduce Miranda, an efficient reputation-based mechanism that detects and isolates active malicious mixes, performing active dropping or delyaing attacks to support the adversary's efforts to deanonymize the users. Miranda combines packet receipts and loop traffic, with a novel approach of examining links between mix nodes, instead of focusing on the mixes themselves. Moreover, in this chapter we present the first systematic analysis using quantitative and composable measure of security against dropping attacks.

4.1 Introduction

In the previous chapter, we presented a modern design of mix network based anonymous communication system, which guarantees resistance against powerful traffic analysis attacks and $(n-1)$ active attacks. In Loopix, the core idea for detecting the active attacks is to deploy a special type of *loop* messages generated by both clients and the infrastructure mix nodes. However, this idea is limited to detecting only aggressive $(n-1)$ attacks, while the mix nodes systematically dropping or delaying single packets can operate undetected. Such active attacks have severe repercussions for privacy and efficiency of mix networks. For example, in the disclosure attack [193] the adversary tries to uncover with whom a target sender is communicating by observing the recipients corresponding to sender's messages and infering the ultimate recipient by identifying the mutually disjoint sets of them. Such disclosure attack in which a rogue mix strategically drops packets from a specific sender can be more effective as it allows the attacker to infer with whom the sender is communicating by observing which recipient received fewer packets than expected. Similarly, the Denial-of-Service (DoS) attacks reduce anonymity and can be used to enhance de-anonymization [3]. As showed by Borisov et al. [3] the system under

a selective DoS attack, in which the adversary affects only the selected parts of the network, becomes much more vulnerable to the deanonymization attacks, than a reliable system under traditional attacks.

Therefore, reliability has a significant impact on the security of the system, and it is important to ensure reliability against adversaries, and not just random failures. In this section, we further study the idea of client-generated loop packets to propose a mechanism which allows detecting malicious mix nodes performing active attacks and enhance reliability of the network.

It is challenging to identify and penalize malicious mixes while retaining strong anonymity and high efficiency. Trivial strategies for detecting malicious mixes are fragile and may become vectors for attacks. Rogue mixes can either hide their involvement or worse, make it seem like honest mixes are unreliable, which leads to their exclusion from the network. The literature on secure electronic elections has been preoccupied with reliable mixing to ensure the integrity of election results by using zero-knowledge proofs [83, 84, 85] 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 networks with randomized partial checking (RPC) [89]. 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 [90].

In this chapter, we revisit the problem of making decryption mix networks robust to malicious mixes performing active attacks. We start in Section 4.2 by performing a theoretical analysis of the impact of selective dropping attacks on the adversary's capabilities to correlate communicating users. Further, we propose *Miranda*, an efficient reputation-based mechanism, that detects and isolates active malicious mixes, which attempt to drop packets to support the adversary's efforts to deanonymize the users. The architectural building blocks behind Miranda, such as *packet receipts* and *loop traffic*, have been studied by previous research (see Section 4.10), but we combine them with a novel approach of examining inter-mix links, instead of focusing on the mixes themselves.

Chaum's original mix network design [49] included a system of signed receipts, which ensure that each mix correctly processed received messages. This

idea inspired a branch of research itself, focusing on the verifiable properties of the mix networks (see Chapter 2). In Chaum's design 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, [49] uses the untraceable return addresses to provide end-to-end receipts for the sender.

Miranda takes advantage of detecting a failure of inter-mix links in addition to the direct detection of corrupt mixes, to disconnect corrupt mixes and ensure that each dropped packet penalizes the adversary. This allows Miranda to mitigate active attacks, without requiring expensive computations, and hence strengthen decryption mix networks and improve their reliability.

Chapter outline: The rest of this chapter is organised as follows. In Section 4.2, we analyze the impact of active attacks on successful deanonymization of communicating parties. In Section 4.3, we explain the system model and define the threat model and security goals. In Section 4.4, we present the high-level overview of Miranda, and Section 4.5 and Section 4.6 detail its core mechanisms, which detect and penalize active attacks. In Section 4.7, we describe the community based protocols which allow to further enhance the detection of malicious mixes. In Section 4.8, we evaluate the security properties of Miranda against active attacks. The comparison between our design and related work is outlined in Section 4.10. Finally, we conclude in Section 4.11.

4.2 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. Therefore, it is important to understand what impact on users deanonymization such attacks may impose. To motivate our work, we first quantify how active attacks threaten anonymity in mix networks. We start by defining a security game. Following it, we present a qualitative and composable measure of security against dropping attacks. To our knowledge, this is the first analysis of such attacks. Our results support the findings of previous works on statistical disclosure attacks [193] and DoS-based attacks [3], arguing that the traffic analysis advantage gained from dropping messages is significant.

4.2.1 Security game

We define a game in which an adversary is trying to correlate communicating sender and recipients, by observing the network and performing packet dropping. Our security game is defined as follows. The challenger chooses a *target mix path* 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 cascade P_x, and observes the system over multiple rounds. Note, that in reality, the adversary might not have such knowledge about S_0, S_1, R_0, R_1 a priori and might have to consider a larger set of potential senders and receipients.

We assume that a set of other clients also communicate using cascade 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 in previous chapter, as $S_0 \rightarrow R_0$ and $S_1 \rightarrow R_1$. Otherwise, if $b = 1$, $S_0 \rightarrow R_1$ and $S_1 \rightarrow R_0$. 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 $b = b'$.

4.2.2 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 received by R_0 and R_1. 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*. The adversary examines how the volume of traffic received by R_0 and R_1 is affected by the active attack, and uses it to increase the probability of correctly guessing b'.

We use a differential privacy metric [55] (described in Section 2.1) to bound the likelihood ratio of the observation (x_{R_0}, x_{R_1}) conditioned on the adversary's guess b' using an $\varepsilon \geq 0$ 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 that the volume of *cover traffic* (the traffic generated by non-target clients) is injected according to the Poisson distribution with parameter λ. We

choose the Poisson distribution since sending can be modeled only by a positive distribution. Moreover, Poisson models have been widely used in computer networks and telecommunications literature [194] 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 Palm's Theorem [195] we know that under suitable conditions a large number of independent multiplexed streams approach a Poisson process as the number of processes grows. Finally, the memoryless property of the Poisson process allows simplifying queuing problems involving Poisson arrivals. However, we highlight that the analysis could be done using other distributions as well.

Now, let us define Y_0, Y_1 as random variables, such that $Y_0 \sim \mathsf{Pois}(\lambda_0)$ and $Y_1 \sim \mathsf{Pois}(\lambda_1)$, 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). We define the following theorem.

Theorem 3. *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:*

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

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

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

Proof. Given the observation $O = (x_{R_0}, x_{R_1})$ we consider two cases conditioned by the events that either $b = 0$, i.e., $S_0 \to R_0$ and $S_1 \to R_1$, or $b = 1$, i.e., $S_0 \to R_1, S_1 \to R_0$. Without the loss of generality, we consider a scenario in which the adversary targets sender S_0. We define a differentially private dependency

$$Pr[Y_0 = x_{R_0}, Y_1 = x_{R_1} - 1 | b = 0] \leq e^{\varepsilon} \Pr[Y_0 = x_{R_0} - 1, Y_1 = x_{R_1} | b = 1] + \delta. \quad (4.1)$$

Thus, we compute

$$\frac{Pr[Y_0 = x_{R_0}, Y_{R_1} = x_{R_1} - 1 | b = 0]}{\Pr[Y_0 = x_{R_0} - 1, Y_{R_1} = x_{R_1} | b = 1]} = \frac{\lambda^{x_{R_0}} e^{-\lambda}}{x_{R_0}!} \frac{\lambda^{x_{R_1}-1} e^{-\lambda}}{(x_{R_1}-1)!} \Big/ \frac{\lambda^{x_{R_0}-1} e^{-\lambda}}{(x_{R_0}-1)!} \frac{\lambda^{x_{R_1}} e^{-\lambda}}{x_{R_1}!} = \frac{x_{R_1}}{x_{R_0}},$$

Given that, we calculate the values of δ, defined as $\delta = \Pr[Y_1 \geq e^{\varepsilon} Y_0]$, using the law of total probability and the cumulative distribution function:

$$\begin{aligned}\Pr[Y_1 \geq e^{\varepsilon} Y_0] &= \sum_{i=1}^{\infty} \Pr[Y_1 \geq e^{\varepsilon} Y_0 | Y_0 = i] \Pr[Y_0 = i] = \sum_{i=1}^{\infty} \Pr[Y_1 \geq e^{\varepsilon} i] \Pr[Y_0 = i] \\ &= \sum_{i=1}^{\infty} \mathsf{CDF}_{Y_1}[e^{\varepsilon} i] \cdot \frac{\lambda^i e^{-\lambda}}{i!}.\end{aligned}$$

□

Next, we also provide a loose, but analytic, bound on δ as a function of ε and λ.

Theorem 4. *The value of δ from Theorem 3 for sufficiently large values of parameter λ can be bound as:*

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

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

$$\begin{aligned}\Pr[Y_1 \geq (1+\varepsilon) Y_0] &= \sum_{i=1}^{\infty} \Pr[Y_1 \geq (1+\varepsilon) Y_0 | Y_0 = i] \Pr[Y_0 = i] \\ &= \sum_{i=1}^{\infty} \Pr[Y_1 \geq (1+\varepsilon) i] \Pr[Y_0 = i].\end{aligned} \tag{4.2}$$

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

$$\begin{aligned}\Pr[Y_1 \geq (1+\varepsilon) Y_0] \leq & \underbrace{\sum_{i=0}^{(1-\frac{\varepsilon}{2})\lambda} \Pr[Y_0 = i] \Pr[Y_1 \geq (1+\varepsilon) i]}_{(I)} \\ & + \underbrace{\sum_{i=(1+\frac{\varepsilon}{2})\lambda}^{\infty} \Pr[Y_0 = i] \Pr[Y_1 \geq (1+\varepsilon) i]}_{(II)} \\ & + \underbrace{\sum_{i=(1-\frac{\varepsilon}{2})\lambda}^{(1+\frac{\varepsilon}{2})\lambda} \Pr[Y_0 = i] \Pr[Y_1 \geq (1+\varepsilon) i]}_{(III)}.\end{aligned} \tag{4.3}$$

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

$$\begin{aligned}\Pr[Y_1 \geq (1+\varepsilon)Y_0] = &\sum_{i=0}^{\left(1-\frac{\varepsilon}{2}\right)\lambda} \Pr[Y_0 = i] + \sum_{i=\left(1+\frac{\varepsilon}{2}\right)\lambda}^{\infty} \Pr[Y_0 = i] \\ &+ \sum_{i=\left(1-\frac{\varepsilon}{2}\right)\lambda}^{\left(1+\frac{\varepsilon}{2}\right)\lambda} \Pr[Y_0 = i]\Pr[Y_1 \geq (1+\varepsilon)i].\end{aligned} \tag{4.4}$$

We note that $\Pr[Y_1 \geq (1+\varepsilon)i]$ in the sum over $i = \{\left(1-\frac{\varepsilon}{2}\right)\lambda, \ldots, \left(1+\frac{\varepsilon}{2}\right)\lambda\}$ can be bounded as

$$\Pr[Y_1 \geq (1+\varepsilon)i] \leq \Pr[Y_1 \geq (1+\varepsilon)\left(1-\frac{\varepsilon}{2}\right)\lambda]. \tag{4.5}$$

Following this, we have

$$\begin{aligned}\Pr[Y_1 \geq (1+\varepsilon)Y_0] = &\Pr[Y_0 \leq \left(1-\frac{\varepsilon}{2}\right)\lambda] + \Pr[Y_0 \geq \left(1+\frac{\varepsilon}{2}\right)\lambda] \\ &+ \Pr[Y_1 \geq (1+\varepsilon)\left(1-\frac{\varepsilon}{2}\right)\lambda] \sum_{i=\left(1-\frac{\varepsilon}{2}\right)\lambda}^{\left(1+\frac{\varepsilon}{2}\right)\lambda} \Pr[Y_0 = i]\end{aligned} \tag{4.6}$$

Since Y_0 is a Poisson distributed variable, and we sum up the probabilities of independent events we can bound the whole sum by 1. Hence,

$$\begin{aligned}Pr[Y_1 \geq (1+\varepsilon)Y_0] \leq &\Pr[Y_0 \leq \left(1-\frac{\varepsilon}{2}\right)\lambda] + \Pr[Y_0 \geq \left(1+\frac{\varepsilon}{2}\right)\lambda] \\ &+ \Pr[Y_1 \geq (1+\varepsilon)\left(1-\frac{\varepsilon}{2}\right)\lambda]\end{aligned} \tag{4.7}$$

Now by applying the Chernoff inequality [57] we can derive a final form of our upper bound for δ[1]:

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

□

Comparison. We compare the above bound (Theorem 4), and the exact calculation of δ from Theorem 3, computed using the importance sampling technique, for

[1]Note, that the above bound can be made even a little bit tighter, by doing two more precise steps in Equation 4.7.

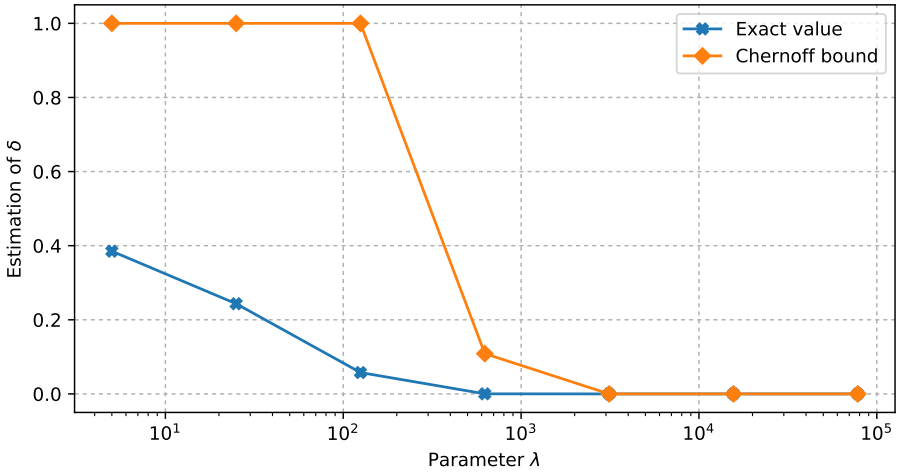


Figure 4.1: The precision of upper bound for δ presented in Theorem 3 for a fixed $\varepsilon = 0.2$.

different values of λ and a fixed leakage $\varepsilon = 0.2$ in Figure 4.1. 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 3 is better for computing a good approximation of leakage δ.

Leakage (ε,δ) for multiple rounds. The adversary can observe the system over multiple rounds. Therefore, we quantify now how much information the adversary can infer in such a scenario. The advantage of the (ε, δ) leakage quantification presented in Theorem 3 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 in round i is defined as $o_i = (x^i_{R_0}, x^i_{R_1})$., we compute

$$\frac{\Pr[(x^1_{R_0}, x^1_{R_1}), \ldots, (x^R_{R_0}, x^R_{R_1}) | b = 0]}{\Pr[(x^1_{R_0}, x^1_{R_1}), \ldots, (x^R_{R_0}, x^R_{R_1}) | b = 1]} = \prod_{i=1}^{R} \frac{\Pr[(x^i_{R_0}, x^i_{R_1}) | b = 0]}{\Pr[(x^i_{R_0}, x^i_{R_1}) | b = 1]} = \prod_{i=1}^{R} \frac{x^i_{R_1}}{x^i_{R_0}} \tag{4.8}$$

From the composition theorem of differential privacy (see Section 2.1) we know that given the value of ε, δ for a single round, the likelihood ratio of multiple observations will follow a similar $(\overline{\varepsilon_R}, \overline{\delta_R})$ relation, with $\overline{\varepsilon_R} = R \cdot \varepsilon$ 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 a significant advantage except with negligible probability. Therefore, we focus on analyzing the average case, for which we simulate several observations $(x^i_{R_0}, {x_{R_1}}^i)$ and compute

the estimator of the average leakage as

$$e^{R\hat{\varepsilon}} = \prod_{i=1}^{R} \frac{x^i_{R_1}}{x^i_{R_0}} \implies \log\left(e^{R\hat{\varepsilon}}\right) = \log\left(\prod_{i=1}^{R} \frac{x^i_{R_1}}{x^i_{R_0}}\right) \implies \hat{\varepsilon} = \frac{1}{R}\sum_{i=1}^{R} \log\left(\frac{x^i_{R_1}}{x^i_{R_0}}\right). \quad (4.9)$$

This allows us to derive the average case value of the leakage which the adversary can gain after multiple concrete observations $(x^i_{R_0}, x^i_{R_1})$. From the law of large numbers [196] we know, that as R grows, the obtained estimator tends closer to its expected value. And thus:

$$\varepsilon_\infty = \lim_{R\to\infty} \hat{\varepsilon} = \mathbb{E}[\log Y/X] \quad \text{for } X, Y \sim \text{Poisson}^+(\lambda) \quad (4.10)$$

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

$$\varepsilon = R \cdot \varepsilon_\infty \quad (4.11)$$

However, we note, that if x_{R_0} or x_{R_1} 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 $\delta_0 = \Pr[x = 0] = e^{-\lambda}$ and $\delta_0 \in [0,1]$. Thus after R observed rounds the probability that any such event has occurred is:

$$\delta = 1 - (1 - \delta_0)^{2R} \quad (4.12)$$

Since R is unbounded, the best estimation we can get for δ is 1. Therefore, we should always compute it directly.

Equation 4.11 and Equation 4.12 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 4.2 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 (we remind that the *cover traffic* here means the traffic of other users, both real and dummy) λ grows, the leakage

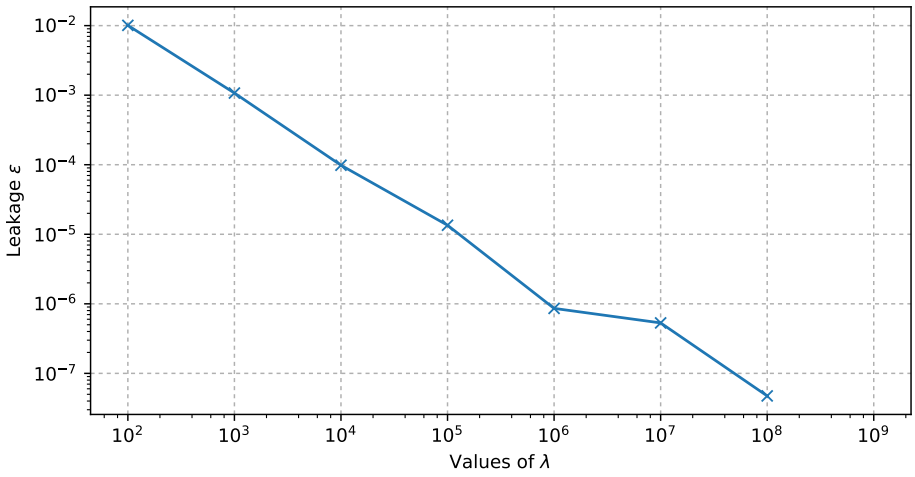


Figure 4.2: The comparison of amounts of leakage ε_∞ for different values of λ.

significantly decreases. For example, for cover traffic rates of $\lambda = 100$, the rate of leakage $\varepsilon_\infty = 10^{-2}$, and thus after $R = 100$ observations we expect a total leakage of $\varepsilon = 1$ (following Equation 4.11). Meanwhile $\delta_0 = e^{-100}$ and overall $\delta < e^{-94}$ (from Equation 4.12) which is tiny.[2]

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 4.2) 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.

In this section, we showed what impact active dropping attacks have on succesfully deanonymization of communicating parties. In next sections, we present a mechanism to detect and exclude malicious nodes from the network.

4.3 System model and security goals

In this section, we outline the general system design, define the threat model, and summarize the security goals of Miranda.

[2] As tiny as breaking a cryptographic key

4.3.1 General System Model

To describe the system, we use the same notation as throughout Chapter 3. We consider an anonymous communication system consisting of a set U ($|\mathsf{U}| = u$) of users communicating over the decryption mix network. As before, we denote by N the set of all mixes building the anonymous network. Depending on the path constraints, the topology may be arranged in separate cascades or a stratified network [164]. We assume that the set of mixes N is fixed (no churn), and we discuss the limitations and challenges of such assumption in Section 4.11. Following [74], we also deploy the idea of secure and decentralized *directory authrorities*. The directory authorities, a set D of semi-trusted servers, maintain a list of available mixes and available links between them. We assume that the set of directory authorities is fixed and known to all clients.

In this chapter, we assume a synchronous mixnet design, in which time is broken into *rounds*. Mixes receive packets within a particular round, denoted by r, decode 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.

Messages traversing the network are end-to-end layer encrypted into a cryptographic packet format by the sender, and the recipient performs the last stage of decryption. Similarly, to the previous chapter, we use the Sphinx cryptographic packet format [177].[3] As before, we also assume the existence of public key infrastructure (PKI) providing an authoritative mapping between mixes and their public keys.

4.3.2 Threat Model

We consider an adversary whose goal is to de-anonymize packets travelling through the mix network. Our adversary acts as a global observer, eavesdropping on all traffic exchanged among the entities in the network, and also, knows the rate of messages that particular users send/receive. We emphasize that this is a non-trivial adversarial advantage. In reality, the adversary might not know users' rate, and therefore might be more limited regarding de-anonymization attacks.

Moreover, all malicious entities in the system collude with the adversary, giving access to their internal states and keys. The adversary may control many participating entities, but we assume a majority of honest mixes and directory servers. We allow arbitrary number of malicious clients but assume that there are also many

[3] However, other packet formats can be used, as long as they fulfil 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*).

honest clients - enough to ensure that any first-mix in a cascade, will receive a 'sufficient' number of messages in most rounds - say, 2ω, where ω is sufficient to ensure reasonable anonymity, for one or few rounds.

In addition, Miranda assumes reliable communication between any pair of honest participants and ignores the time required for computations - hence, also any potential for Miranda-related DoS. In particular, we assume that the adversary cannot arbitrarily drop packets between honest parties nor delay them for longer than a maximal period. This restricted network adversary is weaker than the standard Dolev-Yao model, and in line with more contemporary works such as XFT [197] that assumes honest nodes can eventually communicate synchronously. It allows for more efficient Byzantine fault-tolerance schemes, such as the one we present. In practice, communication failures *will* sometimes occur, we discuss this and other practical challenges in Chapter 6.

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

Similarly, we denote as d the total number of directory authorities. We assume that a number d_m of authorities can be malicious and collude with the adversary or deviate from the protocol, in order to break the security properties. By d_h we denote the number of honest authorities ($d = d_m + d_h$), which follow the protocol truthfully.

4.3.3 Security Goals of Miranda

As defined in Chapter 2 the main goal of a mix network is to hide the correspondence between senders and recipients of the messages in the network. Hence, the Miranda design aims to provide protection which is *indistinguishable from the protection provided by an 'ideal mix'*, i.e., a single mix node which is known to be honest. As we presented in Section 4.2 long term dropping attacks have a significant impact on anonymity through traffic analysis. Therefore, the key goals of Miranda relate to alleviating and discouraging such active attacks. This is achieved through the detection and exclusion of misbehaving mixes. We summarize the protections against active attacks offered by Miranda as follows:

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

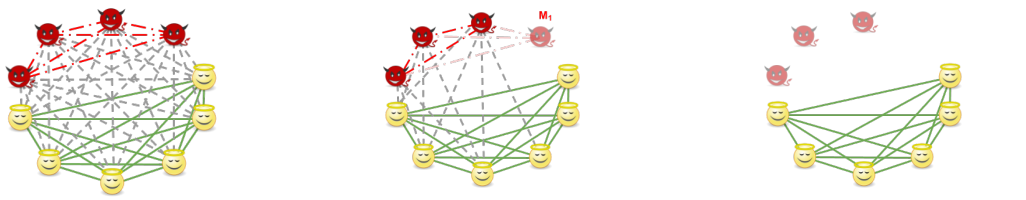
Separation of malicious nodes. Every active attack by a rogue mix results, with a non-negligible probability, in the removal of at least one link connected to the rogue mix - or even the removal of the rogue mix itself.

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.

4.4 The Big Picture

In Miranda, as in other synchronous mixnet designs, time is broken into *rounds*. In addition, rounds are collected into *epochs*, denoted by E, which are used to manage Miranda. The beginning of each epoch includes the announcement of the set of cascades to be used in this epoch, after a selection process that involves avoidance of mixes detected as corrupt, and of links between two mixes, where one or both of the mixes reported a problem. The process of selecting the set of cascades for each epoch, is called the *inter-epoch process*, and is performed by the decentralized directory authorities.

We represent the connectivity of the network as a graph of trust. If two mix nodes trust each other, they share a link in the graph. At the beginning of the system, we assume that this graph is fully connected (see Figure 4.3 (a)). During each epoch, there are multiple rounds where users communicate over the mix network, and both users and mixes report any misbehavior they encounter to the directory authorities. Each active attack – including dropping packets – leads to reduced connectivity for corrupt mixes and reduces their ability to attack, and, eventually, to the detection of corrupt mixes. We refer to the mechanisms that operate during an epoch in order to detect active attacks as *intra-epoch operations*. Miranda disconnects corrupt mixes by carefully gathering reports of their misbehavior, from both clients and mixes, resulting in the removal of links which are misused by the adversary (see Figure 4.3 (b)). The directory authorities process these reports, and, before the beginning of a new epoch, they select a set of cascades available in that epoch. The newly generated cascades will reflect all reported misbehaviors. Namely, cascades exclude links between mixes that were reported. Repeated misbehaviors result in the complete exclusion of the misbehaving mixes from the system. We denote the number of reports which marks a mix as dishonest and causes its exclusion from the network as f and emphasize that f is cumulative over rounds and even epochs. We simply use $f = n_m + 1$, which suffices to ensure that malicious mixes cannot



(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.

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

Figure 4.3: High-level overview of the process of isolating malicious mixes in Miranda.

cause Miranda to exclude honest mixes.[4] However, we find it useful to maintain f as a separate value, to allow the use of larger value for f to account for a number of failures of honest mixes or links between honest mixes, when the Miranda design is adopted by a practical system. On top of disconnecting the faulty links or excluding malicious nodes based on the first-hand evidence, Miranda also applies the *community detection techniques* to enhance the effectiveness of the detection mechanism, see Figure 4.3 (c).

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

4.5 Intra-Epoch Process

In this section, we present the mechanisms that operate during an epoch to deter active attacks, including dropping attacks. We start by describing how active attacks are detected and how this deters malicious behavior. Next, we discuss nodes who refuse to cooperate. Note that in this section, as in the entire Miranda design, we assume reliable communication between any pair of honest participants [198]. As we explain in Section 4.3.2, a practical system deploying Miranda should use a lower-layer protocol to deal with, even severe, packet losses, or introduce a threshold of acceptable losses.

[4] Of course, f can be much higher than n_m. In general, the number of malicious nodes n_m will not be known exactly, and the value of f implemented by the mechanism impacts how efficient the removal of maliocious nodes will be. A precise analysis of which value of f should be selected in practise we leave as a future work.

4.5.1 Message Sending

At the beginning of each epoch, clients acquire the list of all currently available cascades from the directory authorities. Note, that also a Stratfied topology (as define is Section 3.2 in previous chapter) can be represented as a set of cascades. 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 forward packet.

4.5.2 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 [199] statement confirming that a packet p was received by mix M_i. Receipts must be sent and received by the preceding mix within the same round in which packet p was sent.

Generating receipts. We denote a receipt for a single packet p as

$$\mathsf{receipt} \leftarrow \mathsf{Sign}(\mathsf{p} \mid\mid \mathsf{receivedFlag} = 1),$$

where $\mathsf{Sign}(\cdot)$ is a secure digital signature algorithm, and $\mathsf{Verify}(\cdot)$ is its matching verification function.[5] 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 [200] and sign the root of the tree once. Clients' packets are grouped in a single Merkle tree T_C and packets from mix M_i are grouped in a Merkle tree T_{M_i}. Mixes then generate two types of receipts: (1) receipts for clients

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

and (2) aggregated receipts for mixes. Each client receives a receipt for each message she sends. Client receipts are of the form: $\mathsf{receipt} = (\sigma_C, \Gamma_p, r)$, where: σ_C is the signed root of T_C, Γ_p is the appropriate information needed to verify that packet p appears in T_C, and r is the round number. Similarly, each mix, except the last one, receives a receipt in response to all the packets it forwarded in the last round. However, unlike client receipts, mixes expect back a *single* aggregated receipt for all the packets they sent to a specific mix. An aggregated receipt is in the form of: $\mathsf{receipt} = (\sigma_i, r)$, where: σ_i denotes the signed root of T_{M_i} and r is the round number. Since mixes know which packets they forwarded to a particular mix, they can recreate the Merkle tree and verify the correctness of the signed tree root using a single receipt. Once a mix sent an aggregated receipt, it expects back a signed confirmation on that aggregated receipt, attesting that it was delivered correctly. Mixes record the receipts and confirmations to prove later that they behaved honestly in the mixing operation.

Lack of a receipt. If a mix does not receive an aggregated receipt or does not receive a signed confirmation on an aggregated receipt it sent within the expected time slot, the mix disconnects from the misbehaving mix.[6] 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 [201], however, to our knowledge, this approach was not yet applied to the problem of mix network reliability.

Anonymity loves company [191]. Note, however, that this design may fail even against an attacker who does not control any mix, if a cascade receives less than the *minimal anonymity set size* ω. We could ignore this as a very unlikely event, however, Miranda ensures anonymity also in this case - when the first mix is honest. Namely, if the first mix receives less than ω messages in a round, it would not forward any of them and respond with a special 'under-ω receipt' explaining this failure. To prevent potential abuse of this mechanism by a corrupt first mix, which receives over ω messages yet responds with under-ω receipt, these receipts are shared with the directories, allowing them to detect such attacks.

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

4.5.3 Loop Messages: Detect Stealthy Attacks

In a *stealthy active attack*, a mix drops a message - yet sends a receipt as if it forwarded the message. To deter such attacks, clients periodically, yet randomly, *send loop messages to themselves* (see Section 3.4.2). In order to construct a *loop message*, the sender S, chooses a unique random bit-string K_S. Loop messages are encoded in the same manner as the regular messages and sent through the same cascade C selected for the epoch, making them *indistinguishable* from other messages at any stage of their routing. The *loop message* is encapsulated into the packet format as follows:

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

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

If a loop message fails to complete the loop back, this means that one of the cascade's mixes misbehaved. The sender S queries all the mixes in the cascade for evidence whether they have received, processed and forwarded the loop packet. This allows S to isolate the cascade's problematic link or misbehaving mix which caused the packet to be dropped. S then reports the isolated link or mix to the directory authorities and receives a signed confirmation on her report. This confirmation states that the link will no longer be used to construct future cascades. We detail the querying and isolation process in Section 4.5.3.1.

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

This may seem to only ensure α in the context of the messages that Alice *sends* but not against an attack on messages *sent to* Alice. However, notice that if

a corrupt mix M_i drops messages sent to Alice by an honest sender Bob, then M_i faces the same risk of detection - by Bob.

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

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

4.5.3.1 Isolating corrupt mixes with loop messages

Since clients are both the generators and recipients of the attack-detecting *loop messages*, they know exactly during which round r the loop should arrive back. Therefore, if a *loop message* fails to complete the loop back to the sender as expected, the client initiates an *isolation* process, during which it detects and isolates the specific problematic node or link in the cascade. The isolation process starts with the client querying each of the mixes on the cascade to establish whether they received and correctly forwarded the loop packet. During the querying phase, the client first reveals to the respective mixes the packet's *opening value*, in order to prove that it was indeed a loop packet. Next, the client queries the mixes for the receipts they received after they delivered that packet. When clients detect a problematic link or the misbehaving mix, they report it to the directory authorities, along with the necessary proofs that support its claim. This is in fact a broadcasting task in the context of the well-known *reliable broadcast problem* and can be solved accordingly [202]. 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.

When a client asks an honest mix to prove that it received and correctly forwarded a packet, the mix presents the relevant receipt. However, if a mix did not receive this packet, it attests to that by returning an appropriate signed response to the client. If a loop message did not complete the loop because a malicious mix dropped it and did not send a receipt back, the honest preceding mix would have already disconnected from the misbehaving mix. Thus, the honest mix can present the appropriate *disconnection receipt* it received from the directory authorities as an explanation for why the message was not forwarded (see Figure 4.4c).

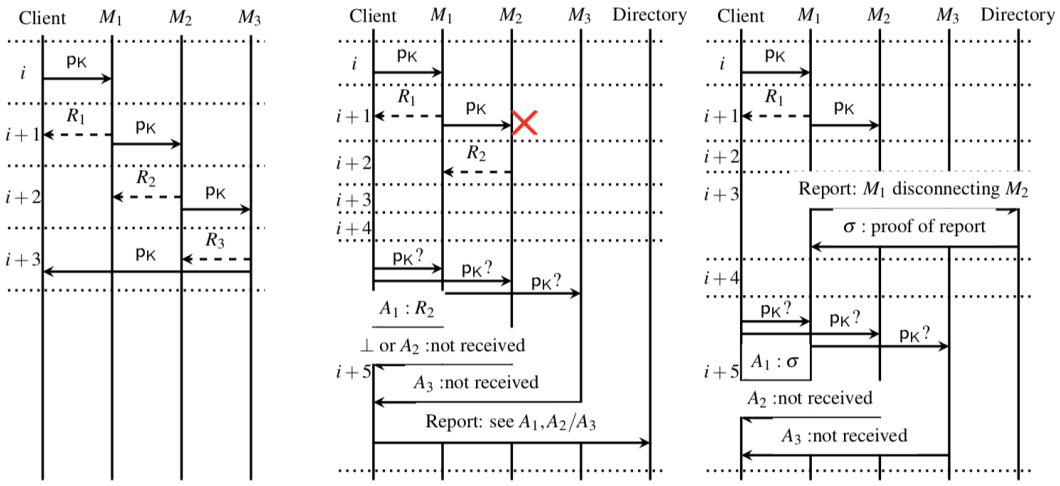
The malicious mix can attempt the following actions, in order to perform an active attack.

Naive dropping. A mix which simply drops a loop packet after sending a receipt to the previous mix can be detected as malicious beyond doubt. When the client that originated the dropped loop packet queries the preceding mix, it presents the receipt received from the malicious mix, proving that the packet was delivered correctly to the malicious node. However, the malicious mix is unable to produce a similar receipt, showing that the packet was received by the subsequent mix, *or* a receipt from the directories proving that it reported disconnection from the subsequent mix. The malicious mix may simply not respond at all to the query. However, the client will still report to the directories, along with the proofs from the previous and following mixes, allowing the directories to resolve the incident (contacting the suspected mix themselves to avoid any possible 'framing') (see Figure 4.4b).

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

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

Delaying packets. A malicious mix can also delay a packet instead of dropping it, so that the honest subsequent mix will drop that packet. However, the honest subsequent mix still sends a receipt back for that packet, which the malicious mix should acknowledge. If the malicious mix acknowledges the receipt, the malicious



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

(b) Example of naive dropping of loop message p_k by M_2, which drops p_k yet sends back a receipt.

(c) Loop packet fails to complete the loop due to non-responding mix.

Figure 4.4: A diagram illustrating loop packets and isolation process.

mix is exposed when the client performs the *isolation process*. The client can obtain a signed receipt proving that the malicious mix received the packet on time, and also the acknowledged receipt from the honest mix that dropped the delayed packet. The latter contains the round number when the packet was dropped, which proves the malicious mix delayed the packet and therefore should be excluded. Otherwise, if the malicious mix refuses to sign the receipt, the honest mix disconnects from the malicious one. Therefore, the delaying attack also causes the mix to either lose a link or to be expelled from the system.

Example. Figure 4.4 illustrates the loop packets and the isolation process. We denote receipt from mix M_i as R_i, and the response as A_i. Figure: (a) presents a succesfull loop trap packet. (b) depicts an example of *naive packet dropping* by M_2. Since p_k did not come back the client queries all mixes during round $i+5$ for the proof of forwarding. M_2 either claims that it did not receive p_k (A_2), thus providing the client a proof that conflicts with the receipt R_2, or M_2 does not cooperate ($\perp$). In both cases the directory authority verifies received A_i's and excludes malicious M_2. (c) shows an example of non-responding mix, where M_1 did not receive receipt from M_2 on round $i+2$ and issues a disconnection in round $i+3$. The client performs the query phase on round $i+5$ and receives the proof of disconnection. The result: M_2 failed to send a receipt to M_1, and thus lost the link to it.

Note that both in (b) and (c) the entire query and report phases occur during round $i+5$, but it could also be spanned across several rounds, as long as it has a

bounded time-frame. For example, if desired, answering the query for p_k could be done in round $i+6$ instead of limiting it to the same round

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

4.5.4 Handling missing receipts

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

We note that a malicious client may trick the directory authorities into performing the isolation process on its behalf repeatedly, against honest mixes. In that case, directory authorities conclude that the mix is honest, since the mix can provide either a receipt for the message forwarded or a disconnection notice. However, this is wasteful for both 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. Furthermore, the clients have to send proofs from the following mix of not having received the packet, which cannot be done if there was no mix failure.

Malicious entry mix. If a first mix does not send a receipt, the client could have simply chosen another cascade; however, this allows malicious mixes to divert traffic from cascades which are not fully malicious, without being penalized, increasing the probability that clients would select other fully malicious cascades instead. To

avoid that, in Miranda, clients *force* the first mix to provide a receipt, by relaying the packet via a trusted ***witness***. A witness is just another mix that relays the packet to the misbehaving first mix. Now, the misbehaving node can no longer refuse to produce a receipt, because the packet arrives from a mix, which allows the isolation process to take place. Note that since a witness sends messages on behalf of clients, the witness *relays* messages without the ω constraint (as if it was a client).

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

4.6 Inter-Epoch Process

In this section, we discuss the inter-epoch operations, taking place toward the end of an epoch; upon its termination, we move to a new epoch. The inter-epoch process selects a new random set of cascades to be used in the coming epoch, avoiding the links reported by the mixes, as well as any mixes detected as corrupt. Until the inter-epoch terminates and the mixes move to the new epoch, the mixes continue with the intra-epoch process as before; the only difference is that newly detected failures, would be 'buffered' and handled only in the following run of the inter-epoch process, to avoid changing the inputs to the inter-epoch process after it has begun. Further in this chapter, we detail the steps of the inter-epoch process.

4.6.1 Filtering Faulty Mixes

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

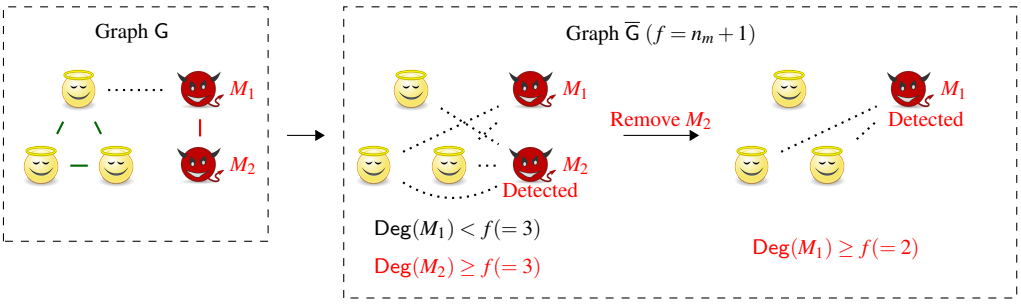


Figure 4.5: An illustration of the simple malicious mix filtering (without community detection).

of rounds, all honest directory authorities have exactly the same set of faulty links.

Note, that only links connected to (one or two) faulty mixes are ever disconnected. Hence, any mix which has more than f links disconnected must be faulty (due to the assumption that $f > n_m$), and hence the directories exclude that mix completely and immediately. Since the directory authorities share exactly the same set of faulty links, it follows that they also agree on exactly the same set of faulty mixes.[7] We call this exclusion process a *simple malicious mix filtering* step. In Section 4.7, we discuss more advanced filtering techniques, based on *community detection*.

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

[7] This is a known problem of distributed systems, therefore we assume that 2/3 of the directory authorities are honest or one of the consensus protocols is implemented.

see whether new malicious mixes can be detected according to the new f value. An illustration of this process is depicted in Figure 4.5.

4.6.2 Cascades Selection Protocol

After all directory authorities have the same view of the mixes and their links, they select and publish a single set of cascades, to be used by all clients during the coming epoch. To allow clients to easily confirm that they use the correct set of cascades, the directory authorities collectively sign the set that they determined for each epoch, using a threshold signature scheme [203, 204]. Hence, each client can simply retrieve the set from any directory authority and validate that it is the correct set (using a single signature-validation operation).

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

The goal of all directory authorities is to select the same set of cascades $\mathsf{C} \subseteq \mathsf{N}^\ell$, where 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. 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., [141]. However, this is a computationally-expensive process, and therefore, we present a much more efficient alternative. Specifically, all directories run exactly the same sampling algorithm and for each sampled cascade validate it using exactly the same legitimate cascade predicate. To ensure that the results obtained by all honest directory authorities are identical, it remains to ensure that they use the same random bits as the seed of the algorithm. To achieve this, while preventing the faulty directory authorities from biasing the choice of the seed bits, we can use a coin-tossing protocol, e.g., [205], among the directory authorities.[8]

[8]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.

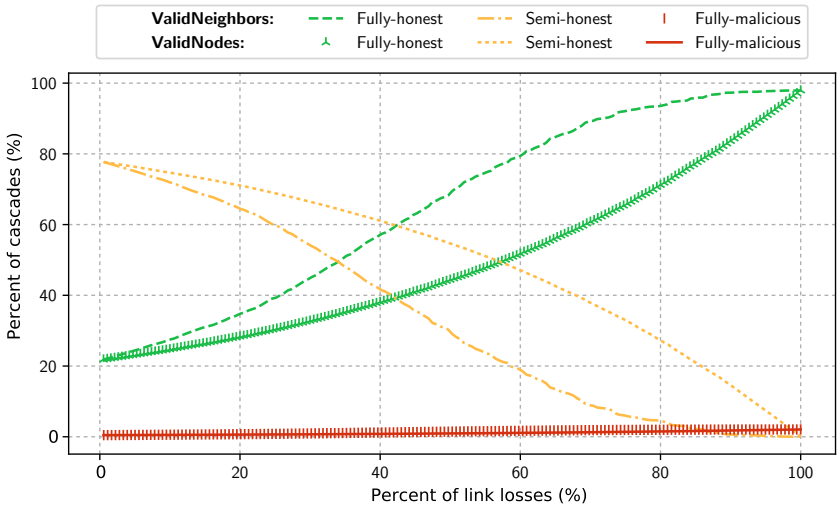


Figure 4.6: Probability of picking cascades as function of link losses, where $l = 4$ and the adversary controls 30% of the mixes.

Legitimate-Cascade Predicates. The path selection protocol can use several constraints, defined by the *legitimate cascade predicates*, to generate a set C of valid cascades. Below, we propose a set of such predicates, which validate whether a cascade is legitimate and can be included in C. Depending on the considered predicate, it can either be co-applied jointly with other ones, to eliminate more undesired cascades, or individually. We propose a set of predicates, defined as $\mathsf{Legit} : \mathsf{N}^{\ell} \rightarrow \{0,1\}$, which given a cascade $c \in \mathsf{N}^{\ell}$ output whether the cascade is valid or not.

- $\mathsf{UniqueInCascade}(c) = \{\forall M_i, M_j \in c : \ i \neq j\}$
  Each mix is used only once in a particular cascade c.
- $\mathsf{NonFaulty}(c) = \{\forall M_i \in c : \ M_i \notin \mathsf{F_M}\}$
  Each mix in cascade c is selected only from the set of non-faulty mixes.
- $\mathsf{OnlyInOneCascade}(c) = \{\forall M_i \in c \wedge \forall \ c' \in \mathsf{C} : \ M_i \notin c'\}$
  Any two cascades should not have a common mix.
- $\mathsf{ValidNeighbor}(c) = \{\forall M_i, M_{i+1} \in c : (M_i, M_{i+1}) \notin \mathsf{F_L}\}$
  For each pair of directly connected mixes in cascade c, this pair should not be listed in the set of faulty links $\mathsf{F_L}$.
- $\mathsf{ValidNodes}(c) = \{\forall M_i, M_j \in c : (M_i, M_j) \notin \mathsf{F_L}\}$
  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 4.6, yet increases the chances of choosing a fully-malicious cascade, as presented further in Section 4.8.2.

4.7 Community-based Attacker Detection

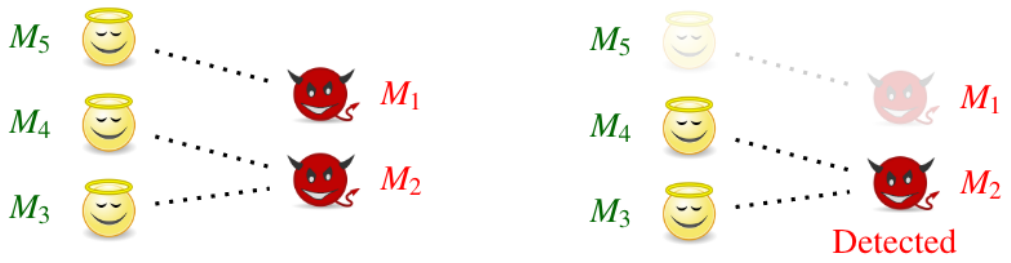
So far, the discussion focused on the core behaviour of Miranda and presented what Miranda can do and how it is done. Interestingly, Miranda's mechanisms open a doorway for advanced techniques, which can significantly improve the detection of malicious mixes. In this section, we discuss several techniques that can leverage Miranda's faulty links identification into a powerful tool against malicious adversaries. Among others, we use community detection techniques. Community detection has been used in previous works to achieve Sybil detection based on social or introduction graphs [131, 206]. However, we assume that the problem of Sybil attacks is solved through other means, such as admission control or resource constraints. Encouragingly, many other techniques can be employed; yet, we hope that the following algorithms will be also useful in other applications where applicable, e.g., where community detection is needed.

4.7.1 Aggresive Pair Removal

From our assumption that honest mixes never fail, we define the following observation.

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

Therefore, a dropped link must be between either an honest mix and a malicious mix or between two malicious mixes. Following this observation, one possible strategy is *aggressive pair removal*, i.e., remove both mixes, if one or both of them report failure of the link connecting them. This strategy seems to provide some benefits - the adversary seems to 'lose more', however it comes at an excess



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

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

Figure 4.7: An illustration of how virtually removing mixes from $\overline{\mathsf{G}}$ can expose malicious mixes. Algorithm 2 refers to the graph in 4.7b as $\overline{\mathsf{G}_1}$, since it is the same graph $\overline{\mathsf{G}}$ as in 4.7a but without M_1 and without M_1's neighbors.

cost of possible exclusion of honest nodes. Therefore, we focus on less aggressive techniques that exclude malicious mixes *without* excluding also honest ones.

4.7.2 Threshold Detection Algorithm

Since the *aggressive removal* of both mixes connected by the failed link from G is not efficient, we adopt the idea of *virtual removal* of the conflicting pair. By *virtually* we mean that virtually removed mixes are not classified as malicious and they are only removed from $\overline{\mathsf{G}}$ for the duration of the algorithm's execution, and not from G nor N. We present the ThresholdDetection technique in Algorithm 1. The algorithm takes as input graph $\overline{\mathsf{G}} = (\overline{\mathsf{V}}, \overline{\mathsf{E}})$, where an edge $(M_i, M_j) \in \overline{\mathsf{E}}$ represents the disconnected link between M_i and M_j. The algorithm starts by invoking the SimpleMaliciousFiltering procedure (described in Section 5.1) on the graph $\overline{\mathsf{G}}$ (line 12).

Next, the algorithm invokes the VirtualPairRemoval procedure on $\overline{\mathsf{G}}$ to *virtually* remove a pair of mixes from $\overline{\mathsf{G}}$ (line 14). Following observation 1, at least one malicious mix was *virtually* removed, thus the *virtual* threshold f' value is decreased by 1 (line 15). We use the f' variable to keep track of the virtually removed malicious mixes and the global f value is decreased only when a malicious mix was actually detected (line 4), and the rest only change the virtual threshold f'. After that, the algorithm invokes the procedure SimpleMaliciousFiltering again on the *updated* $\overline{\mathsf{G}}$ graph, i.e., without the pair of mixes that were virtually removed by the VirtualPairRemoval procedure. The algorithm repeats lines 14-16 as long as there are edges in $\overline{\mathsf{G}}$. For an illustration why the ThresholdDetection algorithm is better than the original *simple malicious mix filtering* see Figure 4.7.

Algorithm 1: ThresholdDetection($\overline{\mathsf{G}} = (\overline{\mathsf{V}}, \overline{\mathsf{E}})$)

```
1  Function SimpleMaliciousFiltering (Ḡ, f′):
2      for every M_i ∈ Ḡ : Deg_Ḡ(M_i) ≥ f′ do
3          M_i is malicious (remove from G, Ḡ, ms);
4          f ← f − 1;
5          f′ ← f′ − 1;
6
7  Function VirtualPairRemoval (Ḡ):
8      Pick an edge (M_i, M_j) ∈ Ē;
9      Remove mixes M_i, M_j from Ḡ.
10
11 f′ ← f;
12 Invoke SimpleMaliciousFiltering(Ḡ);
13 while Ē ≠ ∅ do
14     Invoke VirtualPairRemoval(Ḡ);
15     f′ ← f′ − 1;
16     Invoke SimpleMaliciousFiltering(Ḡ);
```

Next, we improve upon the ThresholdDetection algorithm, while still never removing honest mixes. Our improvement is based on Observation 2 below. But first, we define the following notation, which can be applied to any undirected graph.

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

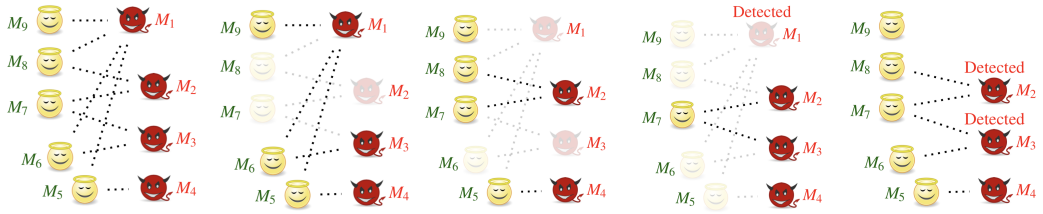
Let us define the graph $\overline{\mathsf{G}_i}$ to be the resulting graph after removing from $\overline{\mathsf{G}}$ the node M_i together with all its *neighbors*, denoted as $\mathsf{N}(M_i)$.

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

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

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

Proof. Assume to the contrary, that there exists a mix M_i such that $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) > n_m - \mu_i$ but M_i is an honest mix. Since there are n_m malicious mixes in N, and μ_i of them are not neighbors of M_i, then the maximum number of malicious mixes



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5

Figure 4.8: A demonstration how Miranda's community detection can significantly improve the detection of malicious mixes using an example graph $\overline{\mathsf{G}}$ and $f = n_m + 1$.

that can be also neighbors of M_i is $n_m - \mu_i$, since M_i is honest. But if $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) > n_m - \mu_i$, then at least one of the neighbors of M_i is also honest, which contradicts the assumption that honest links never fail. Therefore, if $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) > n_m - \mu_i$ then M_i must be a malicious mix. □

Example. Figure 4.7b depicts the graph $\overline{\mathsf{G}_1}$. By observing $\overline{\mathsf{G}_1}$, we know that at least one of the mixes M_2, M_3 are malicious (since they share an edge), therefore, $\mu_i \geq 1$ since we successfully identified a malicious mix which is not in $\{M_1 \cup \mathsf{N}(M_1)\}$. Alternatively, the same argument can be made regarding M_2 and M_4 *instead* of the pair M_2 and M_3. Since after removing M_2, M_4 from $\overline{\mathsf{G}_1}$ there are no edges left in $\overline{\mathsf{G}_1}$, then $\mu_1 = 1$.

Algorithm 2 presents the CommunityDetection algorithm, which leverages Claim 1 to detect malicious mixes.

Notice that the algorithm only examines nodes with a degree larger than 1 (line 3). The reason is that if $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) = 0$ then M_i did not perform an active attack yet, thus it cannot be detected, and if $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) = 1$ then M_i cannot be classified based on its neighbors. Therefore, an execution of the CommunityDetection might not be able to detect all malicious mixes that exposed themselves, e.g., mixes with a degree that equals to 1. If desired, there is always the opportunity to execute the aggressive pair removal technique *after* the CommunityDetection algorithm to potentially remove more malicious mixes (with price of possible removal of an honest mix). Also, randomly picking a pair of mixes that share an edge in $\overline{\mathsf{G}}$ might not always be the optimal strategy. In small graphs, the algorithm can exhaust all possible removal variations, but this is a time-consuming option in large graphs. A more sophisticated picking strategy might yield better results; however, when we experimented with some possible strategies, we did not notice a significant improvement over the random picking strategy.

An illustration of the operation of algorithm 2 is demonstrated in Figure 4.8.

Algorithm 2: CommunityDetection($\overline{\mathsf{G}} = (\overline{\mathsf{V}}, \overline{\mathsf{E}})$)

```
1  n'_m ← n_m;
2  while Ē ≠ ∅ do
3      flag ← 0;
4      for each M_i ∈ V̄ : Deg_Ḡ(M_i) > 1 do
5          Construct Ḡ_i = (V̄_i, Ē_i) from Ḡ;
6          μ_i ← 0;
7          while Ē_i ≠ ∅ do
8              Invoke VirtualPairRemoval(Ḡ_i);
9              μ_i ← μ_i + 1;
10         if Deg_Ḡ(M_i) > n'_m − μ_i then
11             M_i is malicious (remove from G, Ḡ, mixes);
12             n_m ← n_m − 1, n'_m ← n'_m − 1;
13     if Ē ≠ ∅ then
14         Invoke VirtualPairRemoval(Ḡ);
15         n'_m ← n'_m − 1;
```

As depicted in Figure 4.8a $\forall M_i : \mathsf{Deg}_{\overline{\mathcal{G}}}(M_i) < f$, simple malicious mix filtering technique does not detect malicious mixes. However, we can deploy the community based algorithm to improve the detection of malicious nodes. First, we focus on M_2 (Figure 4.8b). When we observe $\overline{\mathcal{G}_2}$, i.e., $\overline{\mathcal{G}}$ after the removal of M_2 and $N(M_2)$, two scenarios are possible. In the first scenario, if M_1 and M_9 are removed first then $\mu_2 = 3$, thus $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 > 1 = n_m - \mu_2$, and therefore M_2 is detected as malicious. In the second scenario, if M_1 and M_6 (or M_5) are removed first then $\mu_2 = 2$, thus $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 \leq 2 = n_m - \mu_2$, and therefore M_2 is *not* detected as malicious (yet). A similar situation occurs with M_3 when observing $\overline{\mathcal{G}_3}$.

On the other hand, when we observe $\overline{\mathcal{G}_6}$ (Figure 4.8c), two malicious mixes can be identified, thus $\mu_6 = 2$. As a result, since $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_6) = 2 \leq 2 = n_m - \mu_6$, M_6 is not classified as malicious (nor should it be). Note that even if M_3 was removed in (b), then $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_6) = 1$ and therefore the algorithm cannot classify it based on its neighbors. The same explanations apply to the rest of the honest mixes.

When we observe $\overline{\mathcal{G}_1}$ (Figure 4.8d), only one malicious mix can be identified, thus $\mu_1 = 1$. As a result, since $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_1) = 4$ is larger than $n_m - \mu_1 = 3$, M_1 is detected as malicious.

If M_2 and M_3 were not detected as malicious as explained in 4.8b, then after the removal of M_1 in 4.8c they will be detected, because the removal of M_1 causes $n_m = 4 \rightarrow n_m = 3$. Since the algorithm runs in a loop, when the algorithm will re-check $\overline{\mathcal{G}_2}$, it will discover that $\mu_2 = 2$ and thus $\mathsf{Deg}_{\overline{\mathcal{G}}}(M_2) = 2 > 1 = n_m - \mu_1$,

which results in removal of M_2. The same goes for M_3. After the removal of M_1, M_2 and M_3, the algorithm cannot classify M_4 as malicious based on its neighbors, since M_4 only dropped one link. However, the algorithm has the option to *aggressively* remove both M_4, M_5.

4.7.3 Community detection based on random walks

In this section, we propose an alternative approach towards community detection, based on random walks inspired by SybilInfer [206]. We define a Markov chain on the graph G as a set of probabilistic transitions for all nodes $M_i \rightarrow M_j$, that we borrow from [206]. We define as $\mathsf{Deg}(M_i)$ the degree of vertice M_i and the probability of transiting from two vertices $M_i \rightarrow M_j$ as

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

and call the matrix of all those transition probabilities Π, while 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 G, as shown in [206]. However, a short random walk, of $\mathsf{O}(\log n)$ steps, will not converge to the stationary distribution for sparser 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. Therefore, since we assume that faulty links can only concern links between the honest and malicious set of nodes, they separate the sub-graph into those two types of mixes. Moreover, even if the adversary attempts to drop his own links between the malicious nodes, he cannot force the algorithm to wrongly exclude the honest mixes, since the attacker is only decreasing the degree $(\cdot)$ of his own malicious nodes. Hence, from the definition of $Pr[M_j|M_i]$, the random walk will tend to move towards the nodes with higher degree, i.e., the honest nodes.

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 $\pi_i^*[j] <$ cutoff the directory authorities remove the link between M_i and M_j thus further pruning the graph used to build cascades.

Summary. The techniques discussed in this section provide Miranda a significant advantage, since malicious mixes can be detected even if they do not pass f. We can 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. Overall, 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. Merely the threat of such techniques is significant in deterring active attacks.

In Section 4.9, we analyze the security of the mechanisms discussed here and evaluate them empirically.

4.8 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.

4.8.1 Resisting Active Attacks

As discussed in Section 4.5, 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.

Delaying packets. A malicious first mix can refuse clients' packets; however, such attack is imprudent, since clients can migrate to other cascades. Furthermore, clients can force the malicious mix to relay their packets, using a witness. Similarly, it is ineffective for the last mix of a cascade to drop *all* packets it receives, since

clients learn through isolation that the dropped loop packets successfully arrived at the last mix. Although clients cannot prove the mix maliciousness, they avoid future cascades containing the malicious mix, including fully malicious cascades.

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

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

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

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

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

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

Aggressive active attacks. In order to de-anonymize the network users, the adversary can choose a more aggressive approach and drop a significant number of packets. For example, in the $(n-1)$ attack [68] 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 [59], where the adversary tries disconnecting target clients. If the adversary cannot directly disconnect a client with a targeted attack, it can disconnect a client by dropping an entire batch of packets where one of them belongs to the client (the adversary simply does not know which). However, it is important to note, that if an adversary can engineer a scenario where a single target packet is injected and mixed with only messages that the adversary controls, *any* mix-based system is vulnerable. Nevertheless, we argue that Miranda inflicts serious penalties on the adversary who attempts to perform an aggressive dropping of packets.

Claim 4. *Miranda deters aggressive active attacks.*

Argument. Aggressive active attacks require the ability to drop many packets. In Miranda, a malicious mix that drops any packet from another mix without sending back a receipt, loses a link (see Section 4.5 and Figure 4.4). Alternatively, if the malicious mix drops packets but does send receipts for these dropped packets, clients can prove that the malicious mix received their (loop) packets and did not forward them, which results in the exclusion of the malicious mix (see Figure 4.4). A malicious entry mix may drop packets from clients, since losing a link to a client is not a serious 'penalty'; but in Miranda, clients then use a *witness mix* (see Section 4.5.4) – forcing the mix to either relay their packets, or - lose a link to a mix or risk discovery, as discussed above.

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

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

Instead, the corrupt mix could send to these clients, or most of them, the special 'under-ω receipt', claiming (incorrectly) that it didn't receive ω messages during this round. However, senders *report* these (rare) under-ω receipts to the directories, who would quickly detect that this mix is corrupt. □

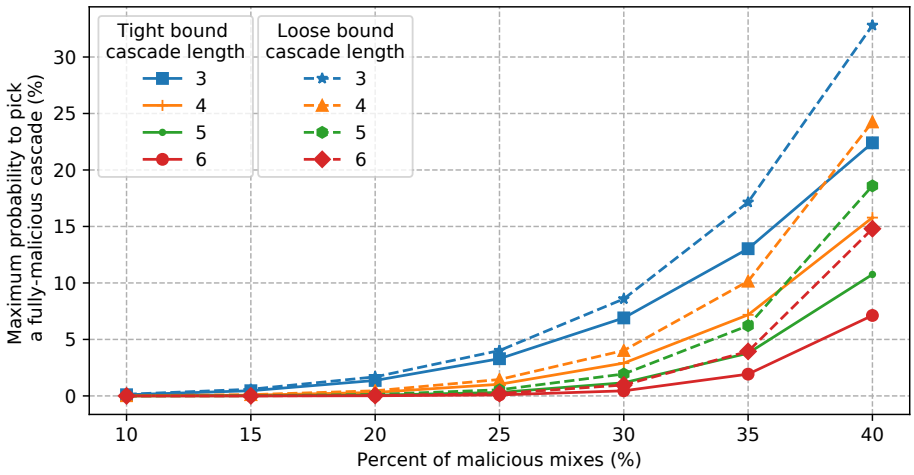


Figure 4.9: The maximum probability of picking a fully malicious cascade as a function of the cascade length and the power of the adversary.

4.8.2 Fully Malicious Cascades Attacks

If the packets are relayed via a *fully malicious cascade*, an adversary can trivially track them. Consequently, adversaries would like to divert as much traffic as possible to the fully malicious cascades. Attackers can try to maximize their chances by: (1) increasing the probability that fully malicious cascades are included in the set 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 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, meaning they cannot exclude any fully honest cascades.[9] However, adversaries are able to disconnect semi-honest cascades by disconnecting semi-honest links and thereby increase the probability of picking a fully malicious cascade. Interestingly, we found that such an attack only slightly increases the chance of selecting a fully malicious cascade – while significantly increasing the chance of selecting a fully honest cascade (see Claim 5). Further, this strategy makes it easier to detect and eliminate sets of connected adversarial domains (see Section 4.7).

Claim 5. *Let C_{Adv} denote a set of fully malicious cascades. The maximum probability to pick a fully malicious cascade during cascades generation process, after*

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

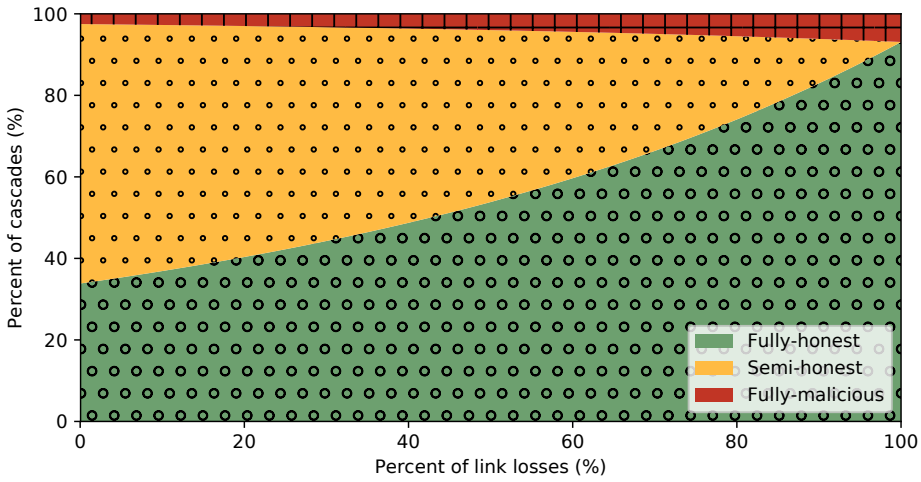


Figure 4.10: The probability of picking particular classes of cascades after each link loss. The parameters of the simulated mix network are $l = 3$, $n = 100$ and $n_m = 30$.

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. 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

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

Figure 4.9 and Figure 4.10 present the probability of picking a fully malicious cascade depending on the number of mixes colluding with the adversary and the percentage of lost links.

Once n_c cascades are generated, the adversary could try to bias the probability of clients choosing a fully malicious cascade. To do so, the adversary can sabotage semi-honest cascades [3] through dropping messages, and in an extreme case, exclude them all. We illustrate in Figure 4.11 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 n_c, the more expensive the attack, and the lower the probability of success.

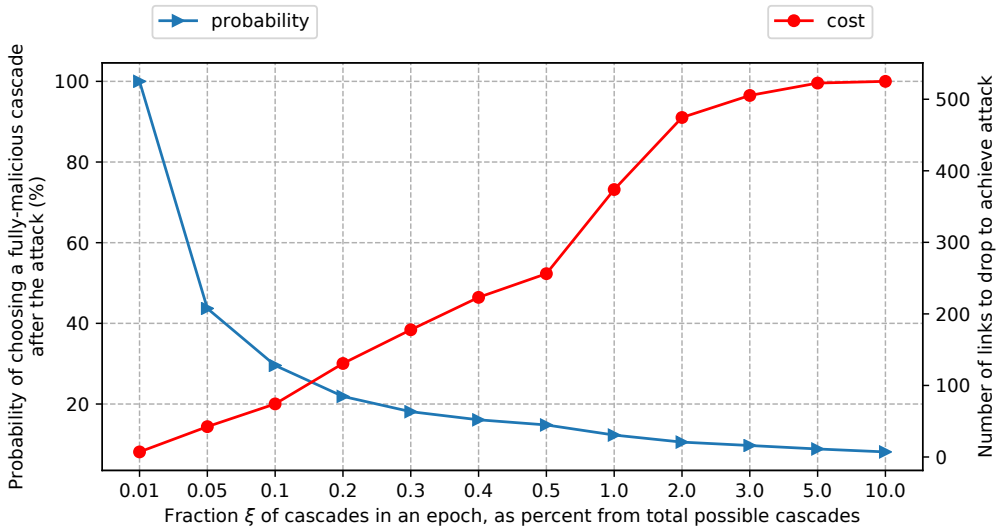


Figure 4.11: The costs and success probability of performing DoS [3] attacks based on the fraction of cascades active in every epoch.

4.8.3 Security of Loop Messages

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

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

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

Argument. It suffices to consider packets sent by non-malicious clients. When a non-last mix receives such packets, it does not know the destination. Furthermore, as described in section 4.5.3, loop packets are sent by non-malicious clients according to the rate defined by α of genuine traffic and are bitwise indistinguishable from genuine packets. Hence, even if the mix would know the identity of the sender, e.g., by being the first mix, the packet can still be a loop message with probability at least α. □

Note that a malicious non-last mix that drops a loop message, yet sends a receipt for it and remains connected to the next mix, would be proven malicious and excluded

from the network. On the other hand, if such mix does not send a receipt, then it loses a link.

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

However, dropping of messages by the last mix can also be done against the *ideal mix* (see Section 4.3.3), e.g., by a man-in-the-middle attacker. In fact, similar correlation attacks can be performed even without dropping packets, if clients have specific sending patterns. Therefore, mitigating this attack is beyond Miranda goals, and should be handled by the applications adopting Miranda.[10]

4.9 Evaluation of Community Detection Techniques

The discussion in Section 4.7 presented several community detection techniques to leverage Miranda's reported links information into a detection tool that removes malicious mixes from the system. We now argue that the contribution of these mechanisms is both important and secure.

4.9.1 Community detection using Threshold Detection and Virtual Pair Removal

We implemented the ThresholdDetection (Algorithm 1) and CommunityDetection (Algorithm 2) algorithms described in Section 4.7, and evaluated them as follows. We generated a complete graph of $n = 100$ mixes where $n_m = 33$ of them are malicious. We modeled a random adversary by randomly dropping a fraction of the semi-honest links, making sure that any mix does not drop more than or equal to $f = n_m + 1$ links. Figure 4.12 demonstrates the effectiveness of the algorithms.

[10]For example, Loopix, presented in Chapter 3, uses fixed sending rate (thus, foiling the attack). A concerned client can simply make sure to send additional loop packets in every round where no genuine traffic is relayed.

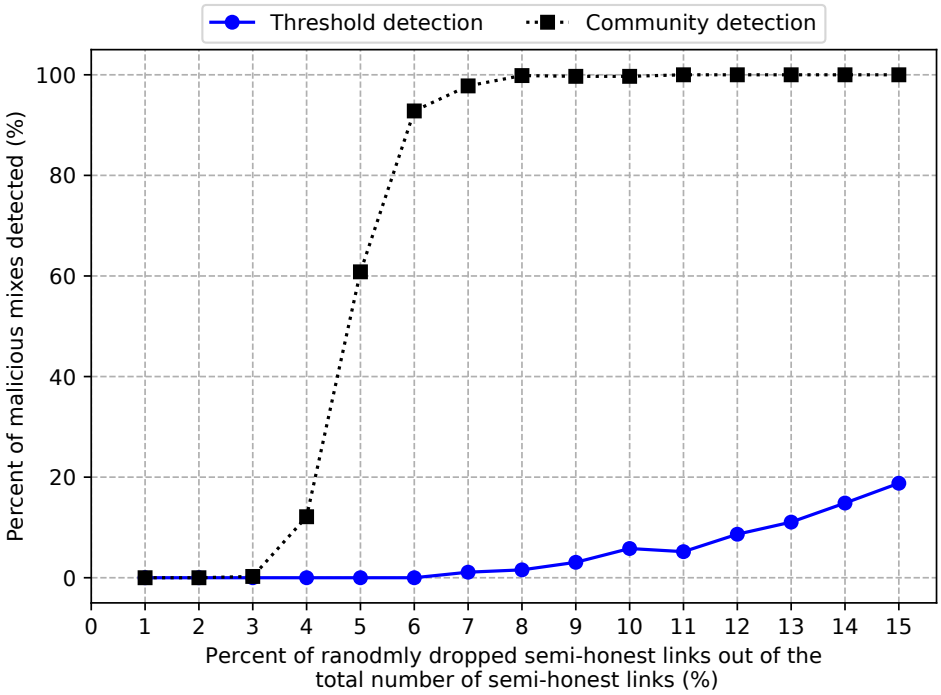


Figure 4.12: The effect of using community detection against malicious mixes.

The ThresholdDetection algorithm starts to become effective when roughly 10% of the semi-honest links are reported and improves as the number of reports increases. In comparison, the CommunityDetection algorithm presents significantly better results, starting when 4% of the semi-honest links are dropped and after 8% the algorithm is able to expose all possible malicious mixes. Considering that the CommunityDetection algorithm can only operate on malicious mixes that dropped more than one link, these results show that the algorithm effectively mitigates the non-strategic adversary.

4.9.1.1 Security Analysis

In essence, both the ThresholdDetection algorithm and the CommunityDetection algorithm do the same thing: they both remove malicious mixes from the system. Therefore, the only way for a strategic adversary to abuse these algorithms is to strategically drop links in a way that causes these algorithms to wrongfully remove honest mixes from the system, due to misclassification of honest mixes as malicious. We now argue that the ThresholdDetection and CommunityDetection algorithms are secured against such attack.

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

Proof. Assume to the contrary that there exists an honest mix $M_i \in \overline{\mathsf{G}}$ that satisfies

$\mathsf{Deg}(M_i) \geq f$. However, if this is the case, then $\mathsf{Deg}_{\overline{\mathsf{G}}}(M_i) \geq f$, which implies $\mathsf{Deg}_{\mathsf{G}}(M_i) \leq n - f \leq n_h - 1$, which means that at least one honest mix disconnected from M_i, contradicting the assumption that honest links never fail. □

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

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

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

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

4.9.2 Community detection based on random walks

We also empirically evaluate the community detection approach based on random walks presented in Section 4.7.3 through simulations. Given a fraction of reported faulty links, we apply our 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 4.13, 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

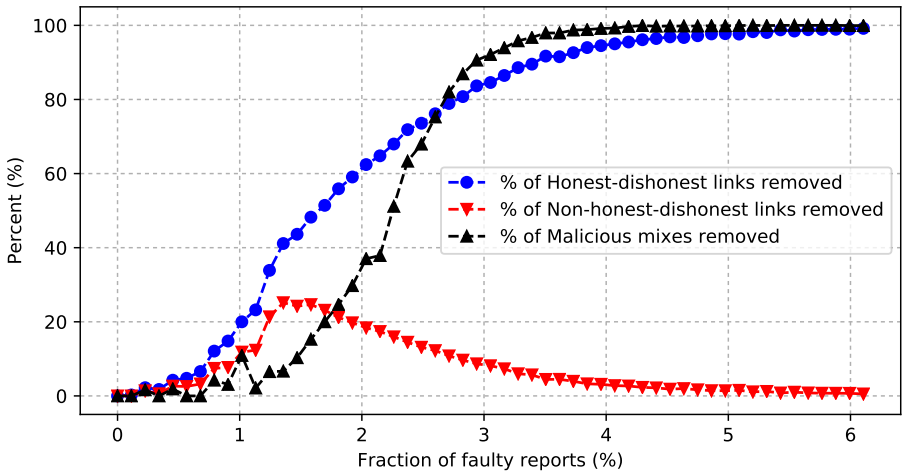


Figure 4.13: Effect of the community detection mechanism to detect semi-honest links.

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. 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 Section 4.2 where each observation of the attacker yields an $\varepsilon_\infty = 10^{-2}$, the attacker has a total attack budget of $\varepsilon = 2.70$ 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 $\varepsilon = 27$ across all users.

4.10 Discussion

The architectural building blocks behind Miranda, such as packet receipts and trap messages, have been studied by previous research. In the work by Dingledine et al. [207], receipts are used to verify a mix failure and rank their reputation in order

to identify the reliable mixes and use them for building cascades. The proposed design uses a set of trusted global witnesses to prove the misbehaviour of a mix. If a mix fails to provide a receipt for any packet, the previous mix enlists the witnesses, which try to send the packet and obtain a receipt. Witnesses are the key part of the design and have to be engaged in every verification of a failure claim, which leads to a trust and performance bottleneck. In comparison, Miranda does not depend on the witnesses, and a single one is just used to enhance the design. Moreover, in this design, a failure is attributed to a single mix in a cascade, which allows the adversary to easily obtain a 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 by Dingledine et al. in [208] 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. Such strong penalty for dropping packets is imposed due to the fact that in contrast to Miranda, this design assumes that honest mixes can accidentally report each other. In order to detect misbehaviours of malicious nodes, the nodes send *test messages* and verify later via a snapshot from the last mix, whether it was successfully delivered. Since the test messages are indistinguishable, dishonest mixes risk being caught if they drop any message. However, the penalty for dropping is very strong – if a single mix drops any message, the whole cascade is failed. Therefore, because a single mix's behaviour 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.

Similarly, the idea of using trap messages to test the reliability of the network was discussed in many works. The original DC-network paper [91] suggested using *trap* messages, which include a safety contestable bit, to detect message disruption. In contrast, the flash mixing [79] technique, which was later proved to be broken [80], 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 [182] mechanism uses heartbeat *loop* messages to detect the (n-1) attacks [68]. 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. Similarly, Mixmaster [73] and Mixminion [74] employed an infrastructure of *pingers* [209], 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. The Atom [150] design as well 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.

Miranda combines packet receipts and loop messages with a novel approach of examining inter-mix links, instead of focusing on the mixes themselves. Thanks to that any malicious behaviour in the network are detected and penalized, hence the adversary cannot silently attack the system and blame honest mixes.

4.11 Conclusion

In this chapter, 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. Our mechanism achieves much better efficiency than previous designs, but at the same time quickly detects and mitigates active adversaries. We deploy previously studied techniques such as packet receipts and loop traffic alongside novel techniques to ensure that each dropped packet penalizes the adversary. We also take a new approach of focusing on problematic links between mixes, instead of mixes themselves. The combination of packet receipts, link disconnection, and loop messages, forces malicious mixes to immediately loose links when they perform active attacks, and in result separates the malicious mix nodes from the honest ones. Further, we also investigated how community detection enhances our mechanism effectively. While our mechanism allows to effectively detect and exclude malicious nodes, it raises also practical open problems which we discuss in Chapter 6. The overall contribution presented in this chapter is an efficient and scalable detection and mitigation of active attacks.

Part II

Applications of mix networks

Chapter 5

Private Notification Service using Mix networks

In this chapter, we introduce AnNotify, a private publish-subscribe notification system, leveraging an anonymous communication network. Simple and fast cryptographic techniques allow AnNotify to be efficient and scale to millions of users while providing rigorous privacy guarantees. Thus, PIR schemes inspired by the AnNotify design can be a great competition to those proposed so far.

5.1 Introduction

In Part I of this thesis, we introduce a novel design of mix network based anonymous communication system and a mechanism which allows mitigating active attacks performed by malicious mix servers. In this part of our work, we study the applications of the mix network systems. The main use-case for the deployment of mix networks is messaging, including emails and instant messaging. However, the proposed designs are not limited to only such use and can be also integrated into other systems, which we discuss in the next chapters.

In this chapter, we show that mix networks may be important building blocks to implement privacy-friendly schemes, such as notifications, that are not entirely related to messaging per se. A number of on-line applications require timely notifications. Mail delivery protocols notify users when a new email can be retrieved, social networking and instant messaging applications send updates of presence, and broadcast notifications carry updates of DNS or cached web records. Traditionally, notification services provide no privacy *vis-à-vis* the notification service itself, that can observe the routing of notifications from the publisher of the event to the sub-

scriber. The fact, that particular consumers are subscribed to a particular publisher or larger groups of publishers can reveal sensitive private information about them. Thus, the privacy-preserving systems, such as anonymous communication systems, or private presence systems [189], rely on private notifications: an adversary should not be able to observe what events a user subscribes to. In this chapter we present AnNotify, a private notification service, leveraging an anonymous communication system, based on simple cryptographic constructions. The AnNotify system is designed for efficiency. Subscribers only retrieve small parts of the event database, to which we refer to as *shards*. Simple and fast cryptographic techniques, allow AnNotify to scale well, while providing rigorous privacy guarantees.

AnNotify has numerous applications. Some only require private notification to signal the availability of a service or a peer (e.g., in instant-messaging systems), or events such as alerts. Other applications, e.g., blacklists, require public notifications with multiple subscribers. Broadcast notifications may signal when a cached value changes - this is especially important for privacy-preserving storage mechanisms such as Oblivious RAM [210, 211] and PIR [139], where each access involves significant overhead. Beyond these, the broadcast notifications can improve the privacy of web and DNS caches, and significantly improve the performance of such caches when they are queried over anonymizing networks such as Tor, see [212, 213, 214].

Chapter outline. This chapter is organized as follows. In Section 5.2 we outline the system model by defining the threat model and security goals, and present the high-level overview of AnNotify. We detail the AnNotify design in Section 5.3. In Section 5.4, we formulate the security definition of a private notification system, define the key security theorems of AnNotify and present its security analysis. In Section 5.5 and Section 5.6, we discuss the costs of AnNotify and compare it to PIR / DP5 [189]. We then discuss possible extensions of AnNotify in Section 5.7. In Section 5.8 we propose a set of applications for our design. Finally, we conclude in Section 5.9.

5.2 System model and Goals

AnNotify is a service connecting *notification publishers* with specific *notification subscribers* that *query* for notifications. We start this section by presenting the high-level overview of the idea. Next, we define both the threat model and security goals of our system. We describe the system for a single subscriber per notification first and extend it later to broadcast to multiple subscribers.

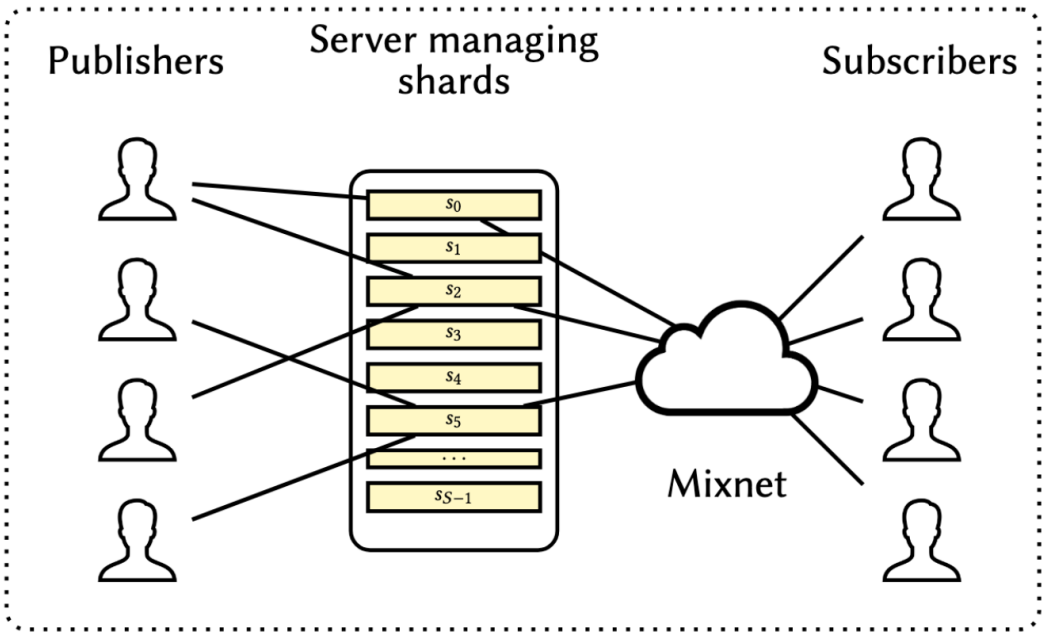


Figure 5.1: The AnNotify architecture.

5.2.1 High-level overview

The AnNotify system consists of multiple **shards** that are managed by a single untrusted server. Shards store information about the presence of the notifications uploaded by the publishers, which subscribers can then query from the system. AnNotify operates in *epochs*. Each epoch publishers, who want to notify the subscriber, connect directly to the system to upload the notifications, whereas the subscribers, in order to subscribe or query for notifications, connect with the servers through the anonymous channels, as illustrated in Figure 5.1. AnNotify uses anonymous channels for communications, and leverages them to increase the efficiency of private queries from a database of notifications. We consider these channels to be perfect, namely to hide all meta-data about senders and receivers of messages, and also the length of messages, as for example Loopix (see Chapter 3).

5.2.2 Security Goals

The AnNotify system provides a number of privacy properties:

Subscriber privacy. Third parties, including the notifier and the infrastructure, cannot tell whether a subscriber sought a notification from a particular publisher.

Epoch unlinkability. An adversary cannot tell whether queries across epochs were initiated by the same subscriber or concern the same notification.

Broadcast privacy. When multiple subscribers are authorized to receive the same notification, corrupt subscribers cannot discover that other honest subscribers are subscribed to the same notification as they are.

5.2.3 Threat Model

The AnNotify design assumes a global passive adversary, who may observe any part or the whole network and tries to infer the relationships between publishers and subscribers. Additionally, we assume that all servers that manage shards may be malicious and work with the adversary. Moreover, AnNotify considers that a fraction of users are malicious: they collude with the eavesdropping adversary, servers or other users to try to break the privacy properties of the system or reveal some information about other users. However, we assume that a large number of concurrent AnNotify users (publishers and subscribers) are honest, and follow the protocol faithfully. We also assume, that the adversary has a partial knowledge about the relationships among publishers and subscribers , and that the adversary may chose to some extent which honest users participate in the protocols at different times. We justify those assumptions further in the paper.

All communications among the requesting subscribers and the servers go through a secure anonymity network. We assume that this system is immune to traffic analysis. Namely, from the point of view of the adversary, it provides a perfect secret permutation between its input and output messages.

5.3 The Design of AnNotify

In this section, we present the detailed description of AnNotify. We first start with sketching the straw-man design based on trivial Private Information Retrieval (PIR), which allows a client to retrieve privately a single record from a remote public database, and argue informally for its security but also its inefficiency. We then present the detailed description of AnNotify. Private information retrieval (PIR) allows a client to retrieve privately a single record from a remote public database. The naive solution retrieves all records from the database, but PIR protocols are more efficient in terms of bandwidth [139, 215, 216].

Straw-man Design

A single server acts as the infrastructure for storing notifications. Publishers and subscribers privately agree on a secret random identifier for a specific notification event. When a publisher wishes to send a notification, she transmits the pre-

arranged random identifier to the server which stores it forever. Subscribers of notifications access the single server, and periodically download the full database of stored notification identifiers, looking for identifiers they recognise as events. This naïve design is secure: since subscribers always download the full database, an adversary at the server cannot distinguish the notification they seek. However, performance is poor: since the database grows continuously, and downloading the full database becomes very expensive. Even using PIR [139], for more efficient private download causes a scalability bottleneck and has performance limitations, as the DP5 presence service [189] illustrates (more in Section 5.6.2). AnNotify provides an efficient and scalable solution to this problem, at the cost of some privacy leakage, which we evaluate carefully.

5.3.1 The AnNotify Protocols

In this section, we detail the design of AnNotify. Figure 3 presents the concrete algorithms of AnNotify, which we discuss informally below.

System setup. We consider a population of u users, distinguished as *publishers* and *subscribers*, using the AnNotify system to exchange notifications. We denote n_S as the number of *shards* used by AnNotify for sharing notifications, and each shard is denoted as $s_i, i \in \{0,\ldots,n_S-1\}$. To increase the capacity and scalability of the system, the shards can be distributed among multiple untrusted servers, however, the number of servers does not impact security. Thus, we consider a single untrusted server managing all shards.

AnNotify uses Bloom filters [56], an efficient data structure used for representing set membership, in order to compress the representation of the shards (for details see Section 2.1). We note that Bloom filters are not used in AnNotify as privacy mechanism, and could be replaced by any other (succinct or not) data representation. As $\mathcal{N}.\mathsf{GenSystem}(u, n_S, \kappa, \Delta)$ we denote the system setup procedure, ran by the server to initialize all parameters of the system, where $\kappa \in 1^*$, $\Delta > 0$ are the security parameters.

A publisher who wishes to send a notification to a subscriber, simply provides them with a secret *channel key* (*ck*) – either directly or derived through a public key cryptographic scheme. We denote the channel establishment procedure as $\mathcal{N}.\mathsf{GenChannel}(\pi)$, where π is the public information.

For publishing and querying notifications clients use a cryptographic Pseudo-Random Function ($\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$) that is indistinguishable from a true random function to a computationally bound adversary not knowing the secret

Algorithm 3: The concrete instantiation of all algorithms of AnNotify.

Function $\mathcal{N}$.GenSystem (u, n_S, κ, Δ):
  Choose packet length l;
  for $i = 0, \ldots, n_S - 1$ **do**
    $s_i^0 \leftarrow []$;
  ▷ σ is the state of the system;
  $\sigma \leftarrow \{s_0^0, s_1^0, \ldots s_{n_S-1}^0\}$;
  $\pi \leftarrow \{u, n_S, \kappa\}$;
  return σ, l, π;

Function $\mathcal{N}$.GenChannel (π):
  $ck \xleftarrow{R} \{0,1\}^\kappa$;
  return ck;

Function $\mathcal{N}$.Notify (ck_i, t):
  $\mu \leftarrow \mathrm{PRF}_{ck_i}(t)$;
  return (i, μ);

Function $\mathcal{N}$.ProcNotify (μ, t, σ):
  $i \leftarrow \mu \mod \pi.n_S$;
  if s_i^t not in σ **then**
    $s_i^t \leftarrow []$;
    $\sigma' \leftarrow \sigma \cup \{s_i^t\}$;
  Add string μ to Bloom filter s_i^t;
  return σ';

Function $\mathcal{N}$.Query (ck_i, t, π):
  $\mu \leftarrow \mathrm{PRF}_{ck_i}(t)$;
  $\phi \leftarrow \mu \mod \pi.n_S$;
  $\phi' \xleftarrow{\$} \{0, \ldots, \pi.n_S\}$;
  return $\{\phi, \phi'\}$;

Function $\mathcal{N}$.ProcQuery (ϕ, t, σ, sk):
  $\rho \leftarrow (\phi, s_\phi^t \text{ from } \sigma)$;
  return ρ;

Function $\mathcal{N}$.ProcResponse (ρ, ck, t):
  $\mu \leftarrow \mathrm{PRF}_{ck}(t)$;
  $\phi \leftarrow \mu \mod \pi.n_S$;
  $\phi'', s \leftarrow \rho$;
  if μ in s and $\phi = \phi''$ **then**
    return True;
  else
    return False;

key [217]. The AnNotify system operates in sequential epochs, like Apres [218], denoted by t for time. For simplicity we assume that the length of all notifications, queries and responses, is always a fixed value ℓ.

Publishing notifications. To publish a notification the publisher runs $\mathcal{N}$.Notify which derives an epoch specific notification identifier ID^t_{ck} for a particular event using a PRF. For each single notification the publisher computes the event notification identifier for epoch t using the shared channel key ck as $\mathsf{ID}^t_{ck} = \mathsf{PRF}_{ck}(t)$. The publisher then computes the index of shard s_i in which the notification should be stored as $i \leftarrow \mathsf{ID}^t_{ck} \bmod S$. Finally, the publisher sends ID^t_{ck} directly to the server managing shard s_i. This process spreads different notifications across shards. The server may optionally perform some authentication and authorization of publishers before accepting to store the notification. Our scheme does not impede this, but details around authenticity are outside the scope of this work.

Storing notifications. The server manages a set of shards, modeled as Bloom filters, for a given time epoch t. Upon receiving a notification ID^t_{ck} at epoch t server runs procedure $\mathcal{N}$.ProcNotify, which adds the notification to a Bloom filter $\mathsf{B}_{i,t}$ for shard s_i, which includes all received notifications for a particular epoch. The server makes all shards available for download in the next epoch.

Querying for notifications. To check for notifications, subscribers repeatedly poll, in every epoch, the server for notifications by downloading the shards of interest via an anonymous network. At the beginning of epoch $t+1$ each subscriber reconstructs the epoch event identifier ID^t_{ck} for the notifications they wish to check for the previous period t by computing $\mathsf{ID}^t_{ck} = \mathsf{PRF}_{ck}(t)$. Next, they recompute the shard identifier $i \leftarrow \mathsf{ID}^t_{ck} \bmod S$, in which ID^t_{ck} might be stored. We denote this querying procedure as $\mathcal{N}$.Query. Alongside the query for the 'real' shard of interest each honest user anonymously sends a 'dummy' indistinguishable and unlinkable query to a random shard. These dummies ensure that no matter what side information is available to the adversary, each honest user contributes some uncertainty to the pattern of queries for the epoch. The notification service runs next the $\mathcal{N}$.ProcQuery in order to process the received queries and returns the obtained results to the subscribers. Each subscriber then *anonymously*, through a mix network, downloads the Bloom filter $\mathsf{B}_{i,t}$ for shard s_i.

Processing the reponse. Upon receiving a response from the server, the subscriber triggers the procedure $\mathcal{N}.\mathsf{ProcResponse}(\rho, ck, t)$, which checks whether ID^t_{ck}

is present in the filter or not. This procedure may yield a *false positive* match, misleading the subscriber into thinking that a particular notification was present when it was not. However, selecting proper Bloom filter parameters relative to the number of notifications allows us to minimize the error probability [219].

5.4 Security of AnNotify

In this section, we first discuss an Indistinguishable-Notification Experiment, a challenge game between an adversary and the system, which we use to measure security. Next, we construe the security definition resulting from it, to quantify the privacy properties guaranteed by the notification systems. Finally, we present the main security theorem of our system and the degree of security obtained for concrete parameters.

5.4.1 Game between the adversary and the AnNotify system

In this section, we describe an Indistinguishable-Notification Experiment (IndNotExp), defined in details in Figure 4, addressing the threats identified in Section 5.2. In this experiment, the adversary Advr observes the system over many epochs. There exists a *target subscriber*, that may be subscribed to one of two publishers (A or B) that are controlled by the adversary. The goal of the adversary is to infer to which publisher a target user is subscribed.

At the beginning of time, the experiment flips a bit b at random, and decides which of the two publishers the *target subscriber* subscribes to. Over multiple epochs the adversary schedules multiple notifications and queries to be executed, and has a full control over which honest publishers notify and which honest subscribers query for their respective notifications. We assume, that at least u_h honest subscribers query every epoch. Advr observes the query patterns of the subscribers, including the *target subscriber* requesting the target notifications, possibly over multiple epochs, and tries to guess b. The threat model captured by the Indisitinguishable-Notification Experiment is very generous to the adversary: Advr has a full visibility into the processing of all notifications and all query requests at all shards of the system for as many epochs as they wish. The adversary is also assumed to know the relationship between all honest publishers-subscriber pairs[1]

[1] It is inevitable to model a private notification system that leaks information. Since the adversary may observe the system for a polynomial number of past epochs she may learn all other mappings except the challenge one.

Algorithm 4: The Indistinguishable-Notification Experiment.

Function ***IndNotExp*** $(\mathcal{N}, \mathsf{Advr}, u, n_S, \kappa, \Delta, u_h)$:
- $(\sigma, \ell, \pi) \leftarrow \mathcal{N}.\mathsf{GenSystem}(u, n_S, \kappa, \Delta)$;
- ▷ Generate two challenge Publishers ;
- $ck_A \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$;
- $ck_B \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$;
- $\mathsf{Advr}(ck_A, ck_B, u, n_S, \kappa, \Delta, \pi)$;
- $b \xleftarrow{\$} \{0,1\}$;
- $ck_T \leftarrow (\text{if } b = 0 \text{ then } ck_A \text{ else } ck_B)$;
- ▷ Generate all other Publishers & Subscribers;
- **for** $i = 0, \ldots, u$ **do**
  - $ck_i \leftarrow \mathcal{N}.\mathsf{GenChannel}(\pi)$;
- ▷ Perform many rounds of the protocols;
- **for** $t = 0, \ldots$ **do**
  - $\Psi_t, \Phi_t \leftarrow \{\}, \{\}$;
  - ▷ Trigger some Publishers;
  - **for** $i \in \{0, \ldots, u\} \cup \{A, B\}$ **do**
    - ▷ Advr chooses notifications;
    - **if** $\mathsf{Advr}(i, t, \text{`Notify?'}) = 1$ **then**
      - $\mu_i \leftarrow \mathcal{N}.\mathsf{Notify}(ck_i, t)$;
      - $\sigma \leftarrow \mathcal{N}.\mathsf{ProcNotify}(\mu_i, t, \sigma)$;
      - $\Psi_t \leftarrow \Psi_t \cup \{(i, \mu_i, \sigma)\}$;
  - ▷ Advr sees all notifications and server state. $\mathsf{Advr}(t, \Psi_t)$;
  - ▷ Trigger at least u_h honest Subscribers;
  - $\mathcal{Q}_t \leftarrow \{\}$;
  - $U_t \leftarrow \mathsf{Advr}(t, u_h, \text{`GetSubscribers?'})$;
  - $U_t \leftarrow U_t \cap \{0, \ldots, n\}$;
  - **if** $|U_t| < u_h$ **then**
    - **return** 0;
  - ▷ Challenge the target Subscriber;
  - **if** $\mathsf{Advr}(t, \text{`TargetQuery?'}) = 1$ **then**
    - $U_t \leftarrow U_t \cap \{T\}$;
  - **for each** $j \in U_t$ **do**
    - $\mathcal{Q}_t \leftarrow \mathcal{Q}_t \cup \mathcal{N}.\mathsf{Query}(ck_j, t, \pi)$;
  - **for each** $\phi_j \in \mathcal{Q}_t$ **do**
    - $\rho_j, \sigma \leftarrow \mathcal{N}.\mathsf{ProcQuery}(\phi_j, t, \sigma)$;
    - $\Phi_t \leftarrow \Phi_t \cup \{(\phi_j, \rho_j, \sigma)\}$;
  - ▷ Advr sees all queries and server state;
  - $\mathsf{Advr}(t, \Phi_t)$;
- **return** $\mathsf{Advr}(\text{`Guess?'}) = b$;

and is given the secrets associated with the notifications of the two potential target notifications, modelling corrupt notifiers or other subscribers in a broadcast group. Figure 4 illustrates the detailed IndNotExp experiment as a game in which the adversary controls, for a number of epochs, notifications ($\mathsf{Advr}(i,t,\text{`Notify?'}) = 1$) and queries ($\mathsf{Advr}(t,u,\text{`GetSubscribers?'})$) from users. The adversary is given all the above information including the challenge notification keys ck_A and ck_B (through invocations to $\mathsf{Advr}(\cdot)$). In r rounds, the adversary may chose to trigger the target subscriber to query by setting $\mathsf{Advr}(t,\text{`TargetQuery?'})$ to 1. Finally, the adversary tries to guess a challenge bit $b$ with $\mathsf{Advr}(\text{`Guess?'})$, i.e., tries to decide which target notification was queried by the target subscriber in the protocol run with full knowledge of the secrets it shares with notifiers. The game returns 1 if the adversary guessed correctly.

Based on the challenge experiment presented, we now define a $\Delta-$private notification system.

Definition 1. *A notification system $\mathcal{N}$ is (u_h,u,Δ)-private if for any PPT adversary* Advr *holds:*

$$\Pr[\mathit{IndNotExp}(\mathcal{N},\mathsf{Advr},u,n_S,\kappa,\Delta,u_h)=1] \leq \frac{1}{2}+\Delta+\text{negl}(\kappa)$$

The probability is taken over all coin tosses, including uniform choice of bit b, and where negl$(\cdot)$ *is a negligible function; the inequality should hold for sufficiently large security parameter κ and depends on the number of epochs r the target subscriber was activated to query. For simplicity, we call such a system Δ-private. Intuitively, Δ defines the advantage of the adversary, in successfully guessing which notification the* target subscriber repeatedly queried*, over a random guess. If the adversary would be guessing randomly, she has a* 50% *chances of a correct guess. Thus, Δ quantifies how much additional information the observed system leaks to* Advr.

This definition ensures that AnNotify provides privacy even when the adversary knows the shared key – allowing notification privacy even when the notifier or another subscriber in a broadcast group, is dishonest and working with the adversary.

5.4.2 The Security of AnNotify

In this section, we present the security theorem showing AnNotify to be a secure Δ-private notification system, as defined in Definition 1 (Section 5.4.1). We highlight, that the presented security theorem is very general, thus is not limited

to the AnNotify system but also can be applied to other systems, which distribute information among many entities and base their security properties on an set of honest participants. Examples of such systems are presented in Section 5.7.

We recall, that S denotes the number of shards, u_h denotes the minimum number of honest subscribers querying in every epoch and r denotes the number of epochs the adversary observes the target subscriber querying for a notification.

In order to quantify the security properties of AnNotify, we want to compute the advantage of the adversary in winning the IndNotExp game, thus the chances to break the privacy of a target subscriber. We start by proving a differentially private [134] security bound ε for the privacy loss in IndNotExp, where the target subscriber only sends a single query (see Section 2.1). Let us first define the following notation

Definition 2. *Let*

$$A = \{(x_A, x_B) : \Pr[X_A = x_A, X_B = x_B | I_A] \leq e^{\varepsilon} \Pr[X_A = x_A, X_B = x_B | I_B]\}$$

We say that $\Pr[X_A, X_B | I_A] \leq e^{\varepsilon} \Pr[X_A, X_B | I_B]$ *holds for* $\varepsilon > 0$ *with probability at least* $1 - \delta$ *to mean that* $\Pr[(X_A, X_B) \in A] \geq 1 - \delta$.

In the following lemma, we quantify all the possible scenarios in which the queries sent by the subscribers are distributed among shards in such a way, that the adversary can easily link the *target subscriber* to the notification.

Lemma 2. *Let* X_A, X_B *denote the query volumes observed by the adversary at shards* s_A, s_B *in a single round assuming that queries map to shards following uniform multinomial distribution, and let* I_A, I_B *define events when a particular challenge notification is queried in the final round. An* (ε, δ)-differential privacy *bound by which:*

$$\Pr[X_A, X_B | I_A] \leq e^{\varepsilon} \Pr[X_A, X_B | I_B]$$

holds for $\varepsilon > 0$ *with probability at least* $1 - \delta$*, where*

$$\delta \leq \exp\left(-\frac{(u_h - 1)}{4S}\right) + \exp\left(-\frac{(u_h - 1)}{2S} \tanh^2\left(\frac{\varepsilon}{2}\right)\right). \tag{5.1}$$

The probabilities are taken over all coin flips of honest notification not observed by the adversary.

Proof. We define as S_A, S_B the events that either shard s_A or s_B was queried. For a mapping function F, which maps the notifications identifiers to the storing shards, such that $F(A) = s_A$, we have

$$\Pr[X_A = x_A, X_B = x_B | I_A] = \Pr[X_A = x_A, X_B = x_B | I_A, S_A]. \tag{5.2}$$

Using Lemma 3 defined below and (5.2) we obtain:

$$\begin{gathered}\Pr[X_A = x_A, X_B = x_B | I_A, S_A] = \Pr[X_A = x_A, X_B = x_B | S_A] \\ \Downarrow \\ \Pr[X_A = x_A, X_B = x_B | I_A] = \Pr[X_A = x_A, X_B = x_B | S_A].\end{gathered} \tag{5.3}$$

Now, we can prove that the events when either notification A or B is requested in the challenge is (ε, δ) - *differentially private*, so the adversary who wants to infer which notification is queried is not able to distinguish this two events with significant probability.

We define $k = x_A + x_B - 1$. The probabilities that either notification A or B is request are denoted as

$$\begin{aligned}\Pr[X_A = x_A, X_B = x_B | S_A] = &\binom{u_h - 1}{k} \left(\frac{2}{S}\right)^k \left(\frac{S-2}{S}\right)^{u_h - k - 1} \\ &\cdot \binom{k}{x_A - 1} \left(\frac{1}{2}\right)^{x_A - 1} \left(\frac{1}{2}\right)^{k - x_A + 1} \\ \Pr[X_A = x_A, X_B = x_B | S_B] = &\binom{u_h - 1}{k} \left(\frac{2}{S}\right)^k \left(\frac{S-2}{S}\right)^{u_h - k - 1} \\ &\cdot \binom{k}{x_B - 1} \left(\frac{1}{2}\right)^{x_B - 1} \left(\frac{1}{2}\right)^{k - x_B + 1}.\end{aligned} \tag{5.4}$$

The above equality results from the binomial distribution[2]. Thus, we have

$$\frac{\Pr[X_A = x_A, X_B = x_B | S_A]}{\Pr[X_A = x_A, X_B = x_B | S_B]} = \frac{x_A}{x_B}. \tag{5.5}$$

We would like to ensure, that $\frac{x_A}{x_B} \leq e^{\varepsilon}$, which implies $x_B \geq e^{-\varepsilon} x_A$.

We remind that the expected value of the binomial distribution $Bin(n, p)$ is $E[X] = np$. Thus, the expected value of queries sent to shard S_A and S_B is $E[X] = \frac{2(u_h - 1)}{S}$. If the number of queries destined to S_A and S_B is smaller than the expected value, the adversary observes very *sparse* hiding set for the target query and has bigger chances to guess correctly if either notification from A or B was queried. Thus, we define $C = \frac{2(u_h - 1)}{S} - \gamma$, where $\gamma = \tau \cdot \frac{2(u_h - 1)}{S}$ and $\tau \in (0, 1)$. We define the value of δ (related to the events when it is easy to distinguish the two observations

[2]The probability distribution function of the binomial distribution $\mathrm{Bin}\left(n, \frac{1}{p}\right)$ is $\Pr[X = k] = \binom{n}{k} \left(\frac{1}{p}\right)^k \left(1 - \frac{1}{p}\right)^{n-k}$.

in our challenge) as below

$$\delta = \underbrace{\Pr[X_A + X_B \leq C]}_{\text{Bin}\left(u_h-1,\frac{2}{S}\right)} + \underbrace{\Pr[X_B \leq e^{-\varepsilon} X_A | X_A + X_B \geq C]}_{\text{Bin}\left(C,\frac{1}{2}\right)} \underbrace{\Pr[X_A + X_B \geq C]}_{\text{Bin}\left(u_h-1,\frac{2}{S}\right)} \tag{5.6}$$

$$\delta = \overset{(I)}{\Pr[X_A + X_B \leq C]} + \overset{(II)}{\Pr[X_B \leq e^{-\varepsilon} X_A \wedge X_A + X_B \geq C]}$$

We recall the Chernoff bound [57] for a random variable X, which is a sum of independent variables with Bernoulli distribution, defined as

$$\Pr[X \leq (1-d)E[X]] \leq e^{\frac{-E[X]d^2}{2}}.$$

First, we estimate the part (I) of equation (5.6)

$$\begin{aligned} \Pr[X_A + X_B \leq C] &\overset{X=X_A+X_B}{=} \Pr\left[X \leq \frac{2(u_h-1)}{S} - \gamma\right] \\ &= \Pr\left[X \leq (1-\tau)\frac{2(u_h-1)}{S}\right] \leq \exp\left(-\frac{(u_h-1)\tau^2}{S}\right). \end{aligned} \tag{5.7}$$

Now, we compute part (II) of equation 5.6 by first deriving an upper bound using the Hoeffding's inequality [220].

$$\begin{aligned} \Pr[X_B \leq e^{-\varepsilon} X_A \wedge X_A + X_B \geq C] &= \sum_{i=C}^{u_h-1} \Pr[X_B \leq e^{-\varepsilon} X_A \wedge X_A + X_B = i] \\ &= \sum_{i=C}^{u_h-1} \Pr[X_B \leq e^{-\varepsilon} X_A | X_A + X_B = i] \Pr[X_A + X_B = i] \\ &\leq \sum_{i=C}^{u_h-1} \Pr[X_B \leq e^{-\varepsilon} X_A | X_A + X_B = C] \Pr[X_A + X_B = i] \\ &= \Pr[X_B \leq e^{-\varepsilon} X_A | X_A + X_B = C] \Pr[X_A + X_B \geq C] \end{aligned} \tag{5.8}$$

Thus, $X_B \leq e^{-\varepsilon}(C - X_B)$, which implies $X_B \leq \frac{e^{-\varepsilon}C}{1+e^{-\varepsilon}}$. Applying the Hoeffding's inequality we obtain the following

$$\begin{aligned} &\Pr[X_B \leq e^{-\varepsilon} X_A, X_A + X_B \geq C] = \Pr[X_B \leq \frac{e^{-\varepsilon}C}{1+e^{-\varepsilon}}] \Pr[X_A + X_B \geq C] \\ &\leq \exp\left(-\frac{(u_h-1)(1-\tau)}{S}\left(1 - 2\frac{e^{-\varepsilon}}{1+e^{-\varepsilon}}\right)^2\right) \cdot (1 - \underbrace{\sum_{k=0}^{C-1} \binom{u_h-1}{k} \left(\frac{2}{S}\right)^k \left(1-\frac{2}{S}\right)^{u_h-1-k}}_{\texttt{CDF}[u_h-1,\frac{2}{S},C-1]}) \\ &\leq \exp\left(-\frac{(u_h-1)(1-\tau)}{S}\left(1 - 2\frac{e^{-\varepsilon}}{1+e^{-\varepsilon}}\right)^2\right). \end{aligned} \tag{5.9}$$

As $\mathtt{CDF}[n,p,x]$ we denote a cumulative distribution function of binomial distribution.

Taking these two equations together, we obtain the following bound for the value of δ for $\tau = \frac{1}{2}$

$$\delta \leq \exp\left(-\frac{(u_h-1)}{4S}\right) + \exp\left(-\frac{(u_h-1)}{2S}\tanh^2\left(\frac{\varepsilon}{2}\right)\right) \tag{5.10}$$

The above bound gives us the estimation of the value of δ, which bounds the probability of very rare events which can blight our differential privacy guarantee. Note, that we have a dependency between δ and ε in this equation, so we can select both values to work the best for us.

The presented estimation is however a crude upper bound, thus we next present the precise computation of the probability value, based on the cumulative distribution function.

$$\begin{aligned}
\Pr[X_B \leq e^{-\varepsilon}X_A \wedge X_A + X_B \geq C] &= \sum_{i=C}^{u_h-1} \Pr[X_B \leq \frac{ie^{-\varepsilon}}{1+e^{-\varepsilon}}] \Pr[X_A + X_B = i] \\
&= \sum_{i=C}^{u_h-1} \left(\sum_{k=0}^{\alpha} \binom{i}{k} \left(\frac{1}{2}\right)^i\right) \binom{u_h-1}{i} \left(\frac{2}{S}\right)^i \left(1-\frac{2}{S}\right)^{u_h-1-i} \\
&= \sum_{i=C}^{u_h-1} \mathtt{CDF}\left[i, \frac{1}{2}, \alpha\right] \binom{u_h-1}{i} \left(\frac{2}{S}\right)^i \left(1-\frac{2}{S}\right)^{u_h-1-i}
\end{aligned}$$

where $\alpha = \frac{ie^{-\varepsilon}}{1+e^{-\varepsilon}}$. Hence, we obtain the following value of δ,

$$\delta = \mathtt{CDF}\left[u_h-1, \frac{2}{S}, C\right] + \sum_{i=C}^{u_h-1} \mathtt{CDF}\left[i, \frac{1}{2}, \alpha\right] \binom{u_h-1}{i} \left(\frac{2}{S}\right)^i \left(1-\frac{2}{S}\right)^{u_h-1-i}. \tag{5.11}$$

We can now show, that we can derive the definition of (ε,δ) - *differential privacy* from the above result. Let us denote $Z = [X_A, X_B|I_A]$. Thus, from the law of total probability we have that

$$\Pr[Z] = \Pr[Z|G]\Pr[G] + \Pr[Z|N]\Pr[N]$$

where as G we denote events, when values drawn for X_A, X_B satisfy the inequality $\Pr[X_A, X_B|I_A] \leq e^{\varepsilon}\Pr[X_A, X_B|I_B]$ and as N we denote otherwise. Thus, since we know, that $\Pr[N] \leq \delta$ and both $\Pr[G]$ and $Pr[Z|N]$ can be upper bounded by 1, using above result we obtain

$$\begin{aligned}
\Pr[Z] &\leq e^{\varepsilon}\Pr[X_A, X_B|I_B] + \delta, \\
\Pr[X_A, X_B|I_A] &\leq e^{\varepsilon}\Pr[X_A, X_B|I_B] + \delta.
\end{aligned}$$

□

Lemma 3. *For random variables X_A, X_B and events I_A, I_B, S_A, S_B defined as in Lemma 2 we have the following dependency*

$$\Pr[X_A, X_B | I_A, S_A] = \Pr[X_A, X_B | S_A].$$

Proof. From conditional probability properties[3] and the fact that $\Pr[S_A] = \frac{1}{2}$, $\Pr[I_A, S_A] > 0$ we can write that

$$\sum_i (\Pr[X_A, X_B | i, S_A] \cdot \Pr[i | S_A]) = \Pr[X_A, X_B | S_A].$$

The sum of the probabilities over requested notifications can be considered as a sum of the probabilities for the notification i which map to shard s_A and those who do not. As N_A we denote a set of notifications whose identifiers map to shard s_A. Following this, we can present the previous equation as

$$\sum_{i \in N_A} \Pr[X_A, X_B | i, S_A] \Pr[i | S_A] + \sum_{i \notin N_A} \Pr[X_A, X_B | i, S_A] \Pr[i | S_A] = \Pr[X_A, X_B | S_A].$$

Because $\Pr[i | S_A] = 0$ for each $i \notin N_A$ we have

$$\Pr[X_A, X_B | I_A, S_A] \cdot \sum_{i \in N_A} \frac{1}{|N_A|} = \Pr[X_A, X_B | S_A]$$
$$\Rightarrow \Pr[X_A, X_B | I_A, S_A] = \Pr[X_A, X_B | S_A].$$

□

Intuitively, in the presented lemma, ε is a measure of a flexible leakage, and δ sums up the probabilities of scenarios in which the adversary is easily winning the challenge game.

Since we know how to quantify δ, we need additionally to compute the amount of leakage due to ε. To derive the adversary advantage for r observed queries we use a generic composition theorem. In the following lemma we derive an overall bound of adversary's advantage, in guessing to whom the target user subscribes, after r rounds when the adversary sees the target subscriber querying for the notification. As $S_{b=0}, S_{b=1}$ we denote the events that the subscriber queries a particular shard, where the target notification was uploaded. As $O_i = (X_A^i, X_B^i)$ we denote the observation of the number of queries observed coming to shard s_A and s_B respectively in round i.

Lemma 4. *Let O_i be an (ε, δ)-differentially private observation in round i, on two private inputs $S_{b=0}$ and $S_{b=1}$, for which $\Pr[O_i | S_{b=0}] \leq e^{\varepsilon} \Pr[O_i | S_{b=1}]$ with probability at least $1 - \delta$.*

[3] $\Pr[A|C,B] \cdot \Pr[C|B] = \Pr[A,C|B] \implies \sum_C (\Pr[A|C,B] \cdot \Pr[C|B]) = \Pr[A|B]$, for the events A, B, C, such that $\Pr[C,B] > 0, \Pr[B] > 0$.

If the adversary Advr *is provided with a set of observations over r rounds denoted as* $\bar{O} = (O_1, \ldots, O_r)$ *resulting from either* $S_{b=0}$ *or* $S_{b=1}$*, and tries to guess the input bit b, she succeeds with probability:*

$$\Pr\left[\mathsf{Advr}(\bar{O}, S_{b=0}, S_{b=1}) = b \,|\, \bar{O}\right] \leq \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right) + r\delta + \mathrm{negl}(\kappa),$$

where
$adv(\bar{O}, S_{b=0}, S_{b=1})$ *denotes the guess of the adversary.*

Proof. From Lemma 2 we know, that a differentially private bound for a single round holds for the probability an adversary observes volumes of the shards resulting from events $S_{b=0}, S_{b=1}$ (with some probability $1-\delta$). From the fact, that the observations in each round are independent we obtain

$$\Pr[O_1, O_2, \ldots, O_r | S_{b=0}] = \prod_{i=1}^{r} \Pr[O_i | S_{b=0}],$$

$$\prod_{i=1}^{r} \Pr[O_i | S_{b=0}] \leq e^{r\varepsilon} \prod_{i=1}^{r} \Pr[O_i | S_{b=1}],$$

On the basis that since $\Pr[S_{b=0}] = \Pr[S_{b=1}] = \frac{1}{2}$

$$\prod_{i=1}^{r} \Pr[O_i | S_{b=0}] \Pr[S_{b=1}] \leq e^{r\varepsilon} \prod_{i=1}^{r} \Pr[O_i | S_{b=1}] \Pr[S_{b=0}],$$

$$\prod_{i=1}^{r} \Pr[S_{b=0} | O_i] \leq e^{r\varepsilon} \prod_{i=1}^{r} \Pr[S_{b=1} | O_i],$$

$$\Pr[S_{b=0} | O_1, \ldots, O_r] \leq e^{r\varepsilon} \Pr[S_{b=1} | O_1, \ldots, O_r]$$

Following this, we obtain

$$\frac{1}{2} + \Delta_\varepsilon \leq e^{r\varepsilon}\left(\frac{1}{2} - \Delta_\varepsilon\right)$$

$$\Delta_\varepsilon \leq \frac{e^{r\varepsilon} - 1}{2\,(e^{r\varepsilon} + 1)} = \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right).$$

Given value of Δ_ε, we can compute the adversary's overall advantage after r rounds. The total value of Δ, which is the adversary's overall advantage of guessing successfully is bounded as

$$\Delta \leq \left(1 - \sum_{i=1}^{r} \delta\right) \cdot \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right) + \sum_{i=1}^{r} \delta. \tag{5.12}$$

since the differential privacy holds in each round with probability $1 - r\delta$ or otherwise, if in at least one round $(r\delta)$ the distribution of query volumes does not guarantee the differential privacy, we can assume that the adversary automatically

can guess correctly. Since $(1 - r\delta) \leq 1$ we loosen the bound of Δ by

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right) + r\delta$$

Value of Δ is the advantage of the adversary in successfully guessing the value of bit b over a random guess. □

Based on the above lemmas we derive the security theorem, proving that AnNotify is a Δ-private notification system.

Security Theorem 1. *The AnNotify system is a Δ-private notification system, for $\Delta > 0$ satisfying the following inequality. For any $\varepsilon > 0$,*

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right) + r\exp\left(-\frac{(u_h - 1)}{4S}\right) + r\exp\left(-\frac{(u_h - 1)}{2S}\tanh^2\left(\frac{\varepsilon}{2}\right)\right)$$

Proof. To prove the main security theorem, and ultimately show that AnNotify is Δ-private, we need to show that the adversary can only win the IndisitinguishableNotification game, showed in Figure 4 with an advantage Δ, defined in Definition 1, over a random guess. We do so by first arguing that the adversary learns nothing new[4] from rounds not including the target subscriber, and then computing the advantage given the information about the rounds when the target subscriber was active.

We proceed through a sequence of hybrid games, with slight modifications over the initial security Definition 1, including the IndNotExp experiment (Game_0). We first note that in the concrete protocols $\mathcal{N}$.Notify and $\mathcal{N}$.Query act on notification IDs generated using a pseudo-random function (PRF) keyed with an unknown key to the adversary and the epoch number ($\mathsf{ID}^t = \mathsf{PRF}_{ck}(t)$). Thus, from the adversaries point of view, the IDs and the shards selection look random and the adversary cannot learn the notification or shard number of any other entity. Hence, we can replace all instances of the first invocation of the PRF by true random functions (Game_1). Thus, the adversary can only distinguish between the original experiment Game_0 and Game_1 with negligible advantage due to the properties of secure PRFs.

In Game_1 the information within each epoch not including the target subscriber is statistically independent from the challenge b. Based on this observation, we define Game_2, that consists only of rounds in which the target subscriber is activated to query. Thus, the advantage of the adversary winning Game_2 is equal to winning Game_1.

In each of the remaining rounds of Game_2 the security definition dictates that a number u_h of honest users (including the target subscriber), query for their sought

[4]Remember that the adversary already is assumed to know the correspondence between honest subscriber-publisher pairs, besides the target query in the challenge round.

notification and a dummy shard.

In Game_2 the adversary can observe the ID^t for all notifications that have been seen in each epoch. However there remain u'_h queries ($u_h \leq u'_h \leq 2u_h$) for which the adversary does not know the corresponding ID^t. These are indistinguishable from a random string, and the corresponding queries are distributed uniformly among the shards S. Thus, we define Game_3 in which we simply remove all notifications and queries for which the adversary knows the ID^t from all epochs – and that does not increase the adversary advantage.

Following this, Game_3 consists of epochs within which the uncertainty of the adversary is whether notification A or notification B was queried (depending on the challenge bit b), and the volumes of at least u_h randomly distributed queries across all shards. Thus, for every epoch, the adversary knowing the secret keys ck_A, ck_B now has to decide on the basis of the query volumes X_A and X_B observed in the shard s_A, s_B corresponding to μ_A and μ_B respectively, what the challenge b was.

We compute the adversary advantage in Game_3 directly. We denote as S_A, S_B the events that the target user queried shards s_A, s_B corresponding to notifications A, B. Lemma 2 then shows that in a single epoch given two known shards and $u_h - 1$ queries to uniformly random shards we can find ε, δ such that for notifications A and B and all query volumes observed by the adversary: $\Pr[X_A, X_B | I_A] \leq e^{\varepsilon} \Pr[X_A, X_B | I_B]$ with probability at least $1 - \delta$. Lemma 4 then concludes the proof by showing this differentially private property can be translated to a concrete adversary advantage Δ gained by observing many epochs. □

Security Theorem 1 presents a bound on Δ that provides insight about the adversary's advantage based on the security parameters of the system. The bound for Δ depends proportionaly on the ratio $\frac{u_h - 1}{S}$ and ε.[5] However, this bound is very loose. A tighter bound on Δ is less elegant.

Lemma 5. *The AnNotify system is a Δ-private notification system for*

$$\Delta \leq \frac{1}{2}\tanh\left(\frac{r\varepsilon}{2}\right) + r\mathsf{CDF}\left[u_h - 1, \frac{2}{S}, C\right] + r\sum_{i=C}^{u_h-1} \mathsf{CDF}\left[i, \frac{1}{2}, \alpha\right] \binom{u_h - 1}{i} \left(\frac{2}{S}\right)^i \left(1 - \frac{2}{S}\right)^{u_h - 1 - i},$$

where $\varepsilon > 0$.

where $\mathsf{CDF}[n, p, x]$ is the cumulative distribution function for a binomially distributed variable. We can compute this bound on Δ using Monte-Carlo integration

[5] Note, that the upper bound on δ in Lemma 2 is constant as long as the ratio $\frac{u_h - 1}{S}$ is constant. In Theorem 1, because δ depends on ε, we obtain a uniform bound for Δ for all values of δ, when ε is fixed.

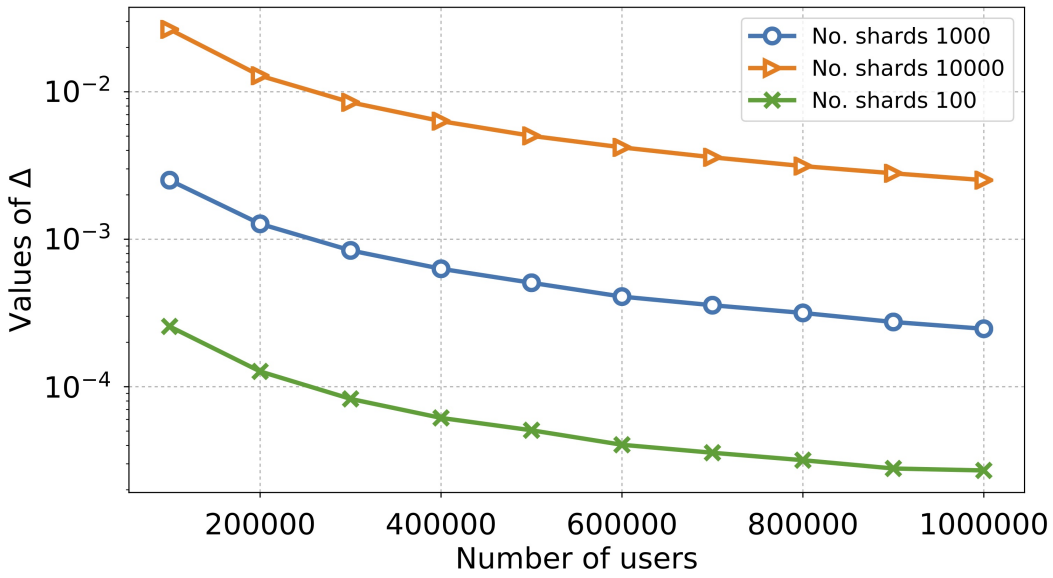


Figure 5.2: The empirical adversary's advantage for a single round, averaged over 10^6 samples, as a function of the number of subscribers and the number of shards. The advantage is presented on a log scale.

though importance sampling.

5.4.3 Empirical adversary advantage

Our security theorems bound the advantage Δ of the adversary through a number of upper bounds and a generic composition theorem. This upper bound is correct but extremely loose: it assumes that in each round the worst possible observation will occur; it discounts totally cases where the adversary observes too few queries to target shards – even though they may hide information; and takes a number of loose upper bounds to yield an insightful expression. To get a more accurate view of $\hat{\Delta}$, the advantage of the adversary, we compute it empirically through sampling.

For fixed parameters u_h and n_S we draw a large number of samples from a Multinomial(θ,n) distribution with parameter vector $\theta = [(n_S - 2)/n_S, 1/n_S, 1/n_S]$ and $n = u_h - 1$, each in effect simulating a single observed epoch. We denote as $\vec{x_A}$ and $\vec{x_B}$ the sample values falling in the second and third bucket respectively. Without loss of generality, we assume that bucket A always gets at least one message. We first compute an empirical $\hat{\delta}$ as the fraction of values in $\vec{x_B}$ that are zero, thus allowing the adversary to perfectly win the IndNotExp experiment. Given the security parameters used in the evaluation this condition is very rare and has never occurred. Next, we estimate $\hat{\varepsilon}$ as the mean leakage the adversary observes for all

samples with positive $\vec{x_B}$:

$$\hat{\varepsilon} = \frac{1}{I} \cdot \sum_i \log \frac{x_A[i]}{x_B[i]},$$

where I denotes the number of samples. This is the log of the Geometric mean of the leakage for each epoch. From the Law of Large numbers [196], we know that for a large number of repeated experiments, the average of the results is close to the expected value, and the more trials we run, the closer the expeted value we are. Hence, the computed value of $\hat{\varepsilon}$ for a large number of samples I quantifies the expected leakage of an observed round. The overall advantage after r epochs can then be computed as:

$$\hat{\Delta} = \tanh(r\hat{\varepsilon}/2)/2 + r\hat{\delta}$$

This empirical advantage is the mean advantage of the adversary after observing a very large number of AnNotify epochs. And given low leakage in every round it is a more accurate depiction of the security of the system under multiple observations than the bound from our theorems. Figure 5.2 depicts the empirically computed adversarial advantage for a single round, over the AnNotify system composed of $10^2, 10^3, 10^4$ shards and a varying number of subscribers querying for notifications.

Further in the work, we use the empirical evaluation to accurately compare security and performance with DP5.

5.4.4 Other security arguments

Our main proof of security of AnNotify concerns the **subscriber privacy** property, under a very strong threat model. We argue informally in this section that other security properties also hold.

The **epoch unlinkability** property ensures that queries in different epochs cannot be linked with each other or a specific subscriber. It is a simple result of the use of keyed pseudo-random function to derive unlikable identifiers within each epoch.

The **broadcast privacy** property ensures that a malicious subscriber, with knowledge of the notification key, is not able to determine whether another query (or subscriber) is querying the same known notification. This property is implied by the very strong INDNOTEXP definition and game. Since the adversary in this game has knowledge of the notification shared key they are exactly in the same position as another subscriber of the same notification, and thus they both enjoy at most the same advantage.

5.5 Analytical Performance Evaluation

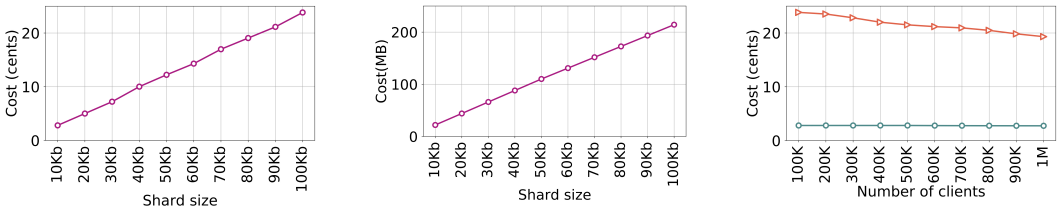
Bandwidth. We evaluate the bandwidth cost of multi-shard AnNotify against the naïve design using a multi-server IT-PIR [139] scheme inspired by DP5 [189]. IT-PIR is a multiple server PIR variant, where each server stores a replicated copy of the database. IT-PIR guarantees perfect privacy, as long as one server is honest, but requires all servers to process each query and operate on the whole database, which increases both the computational and communication costs. A variant of IT-PIR with reduced computational cost was proposed in [221], by allowing some information leakage. However, the key difference between [221] and our work is that AnNotify servers are entirely untrusted and it wholy relies on the anonymity system for privacy.

Let the number of shards in AnNotify be S, and the number of servers in the PIR scheme be S'. Since in AnNotify all shards are of equal size, denoted as l, the number of bits transferred is $nl \cdot m_x$ where n is the number of subscribers that downloaded the Bloom filter and m_x is the cost of using a mix network to transport data (to be fair we assume $m_x = S'$). For the IT-PIR scheme the cost is $nS'\sqrt{v}$, where v is the number of bits in the server's database.

Additionally, since AnNotify may yield false positives, we must consider the bandwidth cost of a subsequent action of a subscriber given that they received a notification, which we denote as a. We intentionally do not specify what this action is, as AnNotify could be used in a variety of applications. Let $k \leq n$ be the number of subscribers who received a notification and f be the error rate of the Bloom filter. Then $h = nf$ subscribers will incorrectly think they have received a notification. Hence the cost of performing actions in AnNotify is $a(k+h)$, whereas in the PIR scheme the cost is ak since no false positives occurs.

The total cost of AnNotify is $nl \cdot m_x + a(k+h) = nl \cdot m_x + a(k+nf)$. The total cost of the PIR scheme is $nS'\sqrt{v} + ak$. We want to estimate the cutoff cost a for AnNotify to be less expensive than a PIR scheme, hence we require $nl \cdot m_x + a(k+nf) < nS'\sqrt{v} + ak$. This gives $a < \frac{S'\sqrt{v} - (l \cdot m_x)}{f}$.

We note that the false positive rate f and the size of the Bloom filter l are related by $f \approx (1/2)^{l \log 2/m}$, where m is the number of messages in the filter, that we assume is approximately N/S where N is the total number of notifications. Similarly, the database in an IT-PIR system would need at least $v = N \log N$ bits to store a list of up to N distinct notifications. Thus, it is preferable to use the AnNotify system over IT-PIR when the cost of an action a is lower than the following threshold: $a < (S'\sqrt{N \log N} - (l \cdot m_x))2^{\frac{lS}{N} \log 2}$.



(a) Yearly cost (cents) per client for different shard sizes

(b) Yearly bandwidth (MB) per client for different shard sizes

(c) Yearly cost (cents) per client using 10Kb (circles) and 100Kb (triangles) shards, for different numbers of clients

Figure 5.3: AnNotify's implementation evaluation summary. The system scales perfectly for the increasing number of clients. Larger shards imply higher bandwidth and cost per client. The cost evaluation was done based on Amazon EC2 `m4.large` instances.

Latency. In the AnNotify system, a notification sent by a publisher in epoch e_i becomes available to a subscriber in epoch e_{i+1}. The time between a notification being sent and when it can be read is $|e| + t$, where t is the round trip time taken by the notification to be routed through the mix network and $|e|$ denotes the server epoch length. Note, that this time t is dependent on the amount of traffic passing through the mix network, and the mix networks flushing mechanism.

Refresh rate, epoch length, cost and privacy. In AnNotify system publishers and subscribers must decide on an epoch length, based on which their notification identifiers will change. There is a clear trade-off: shorter epochs mean shorter waiting times but result in the subscribers requesting more often. Publisher-subscriber epoch lengths are entirely context dependent, for example a social network presence notification system will likely have much shorter publisher-subscriber epoch lengths than a storage system.

5.6 Experimental Evaluation

Three key advantages of AnNotify over previous works [139, 189] are efficiency, extremely low infrastructure cost (even at large scale), and ease of implementation. In this section, we describe a prototype implementation of AnNotify, based on web technologies for the server components, and Tor as an anonymity system. Next, we compare it with DP5.

5.6.1 Implementation & Infrastructure

We implement AnNotify as a web-server that subscribers may easily access through the most popular anonymity network today, Tor [43]. We note, that even though we

use Tor, the anonymous channels might be implemented using other designs, for example Loopix (Chapter 3). We are aware that Tor only provides anonymity properties against a local or limited passive adversary, and thus the experimental system inherits this limitation. Since we are concerned with performance we focus on supporting as many clients as possible, and decreasing the connection time between the client and the server.

Our implementation of AnNotify consists of two servers: a front-end server with whom the clients communicate to download shards, and a back-end server that maintains the Bloom filters. We design AnNotify so that queries are served as requests for a static resource: since those only need to retrieve the Bloom filter corresponding to a previous epoch. The task of the front-end server is simply to serve medium to large static resources; since servers are untrusted, caching and content distribution network may be used to speed this up – and this is a feature of AnNotify. We expect the size of the Bloom filter served to be similar to the size of an image, between several kilobytes to a few megabytes.

To perform a query and retrieve the Bloom filter, AnNotify clients just send an HTTP GET requests to the front-end server. To optionally register a notification, the clients can additionally send the notification identifier for the current epoch as a parameter to the HTTP request. The front-end server immediately responds with the relevant *current* Bloom filter, that is stored as a static file, and forwards the request to the back-end server to update the *next* filter. At the beginning of every epoch, the back-end server sends the *next* Bloom filters, one for each shard, to the front-end server, and the front-end server replaces the *current* Bloom filter with it.

We used Nginx[6] for the front-end server due to its high performance in serving static resources. We implemented the back-end server in Java, relying on Netty[7], a non-blocking I/O (NIO) client-server framework. We relied on Google Guava's implementation of Bloom filter[8]. The front-end implementation simply consists of the Nginx configuration file, and the back-end is 300 lines of Java code.

5.6.2 Performance Evaluation

To evaluate AnNotify, we run an AnNotify server on a single Windows 7 OS, 8GB RAM machine. The back-end and the front-end servers run as two processes. From another machine, we run our client program from several processes to simulate 100K requests in epochs of 5 minutes. We tested the system for shards from 10Kb to

[6]The NGINX Web Server https://www.nginx.com/
[7]The Netty Framework http://netty.io/
[8]Guava: Google Core Libraries for Java https://github.com/google/guava

100Kb. Larger shards imply larger Bloom filters to retrieve and higher bandwidth.

A single machine served 100K clients when the shard size was up to 30Kb. For larger shards we encountered sporadic failures for some clients, and had to add additional servers to handle some shards. The design of AnNotify allows distributing the shards among several machines without overhead. The yearly cost of an Amazon EC2 m4.large instance (in April 2016), which is equivalent to the machine we used, is $603. Dividing the cost of additional machine by 100K clients implies minimal additional cost of less than a single cent per client. Our measurements indicate an additional server is required for each 30Kb increase of the shard size.

We estimated the cost of running AnNotify in the Amazon cloud. The main factor in the cost calculation was the bandwidth that increases linearly as a function of the shard size. However, the bandwidth cost per byte decreases as the system consumes more bandwidth, e.g., for larger shards and for more clients. Figure 5.3 illustrates our costs estimation, extrapolated from measurements using our experimental setup, for a full year of operation in the Amazon cloud. The costs are illustrated in monetary values, on the basis of the cost of an Amazon EC2 m4.large instances. The results show that AnNotify is indeed very efficient, and extremely cheap to operate in the real world. Figure 5.3(a) shows that the yearly cost per client ranges from a few cents (shards of 10Kb) to less than a quarter (shards of 100Kb). Figure 5.3(b) shows the linear growth in the yearly bandwidth used by AnNotify client as a function of a shard size. However, as depicted by Figure 5.3(c), the AnNotify scales perfectly in the number of clients, such that the cost per client even decreases as there are more clients in the system. For a shard of size 10Kb, yearly costs per client is around 3 cents for both 100K and 1 milion users. In comparison, in DP5 the monthly cost per-user for bandwidth is about 0.05 cent, which results in 60 cents per year for 100K users, and around 120 cents for 1 milion users.

5.6.3 Comparison to DP5

Social applications require private presence notifications. Traditional implementations of presence give a central server the social graph of users. Protocols like Apres [218] and DP5 [189] offer privacy-preserving notification services. Apres splits the time into epochs and hides the correlation between the connectivity of the clients in every two epochs. DP5 offers stronger privacy guarantees, however this design uses multi-server IT-PIR to look up other users presence without revealing information about the social graph. In this section, we compare our work with DP5 in our evaluation section.

Compared to the thousands of lines of C++ and Python used to build

DP5 [189], AnNotify was significantly easier to implement and does not require PIR services or Pairing-friendly libraries, however it requires an anonymous channel. Despite being implemented in Java, it efficiently supports a hundred thousand clients, and can be parallelized to scale to millions of clients easily (see Figure 5.3(c)) with significantly lower yearly cost than DP5, of a few cents per client.

Given the different threat models and functionality it is delicate to provide a fair comparison between DP5 and AnNotify calibrated in terms of security. To do so we compare the second phase of DP5, with each user having a single friend, and the status communicated being a single bit notification. Thus, for u users DP5 would have to serve through PIR a database of at least u bits using IT-PIR over ℓ servers, acting as the security parameter. We configure AnNotify to also serve a database of u bits over S shards, using a mix network with path length ℓ. Both ℓ and $S < u$ are the security parameters of AnNotify for a fixed number of users u. We do not use Bloom filters to avoid making assumption on notification utilization, thus presenting a very costly variant of AnNotify.

We consider that either IT-PIR servers or mix servers may be corrupt with a fixed probability f. In that case the advantage of the adversary in DP5 is f^ℓ, namely the probability that all PIR servers are corrupt. For AnNotify the advantage of the adversary is the leakage Δ, that we compute empirically (to get a tight estimate, see appendix 5.4.3), added to the probability f^ℓ that all mix-servers are corrupt.

Bandwidth. Figure 5.4 illustrates the trade-off between security and bandwidth for AnNotify compared to DP5 using the above configuration, for differing security parameters S (shards) and ℓ (mix or PIR servers). We vary $S \in \{10^3, \ldots, 10^8\}$ and $\ell \in \{2, \ldots, 11\}$. The measurements are for one billion notifications ($u = 10^9$) and a fraction $f = 10\%$ of corrupt servers. We observe that AnNotify requires many orders of magnitude (log scale x axis) lower bandwidth per query than DP5 for moderate adversary advantage (e.g., $e^{-5} \ldots e^{-11}$). This advantage is comparable to using $\ell \leq 5$ PIR servers. For each value of S we observe that at first the advantage is dominated by the probability of the mix network failing (for low ℓ) before stabilizing and being dominated by the leakage of AnNotify.

Processing. We implement the DP5 second phase IT-PIR scheme using 64 bit *numpy* matrix multiplication, to compare the CPU costs of AnNotify versus DP5. We note that IT-PIR is CPU bound, while the untrusted servers of AnNotify are purely network bound, since no processing takes place on them aside from serving static shards of data. However, the anonymity network used by AnNotify may become a CPU bottleneck. To estimate this cost we measured the total CPU overhead

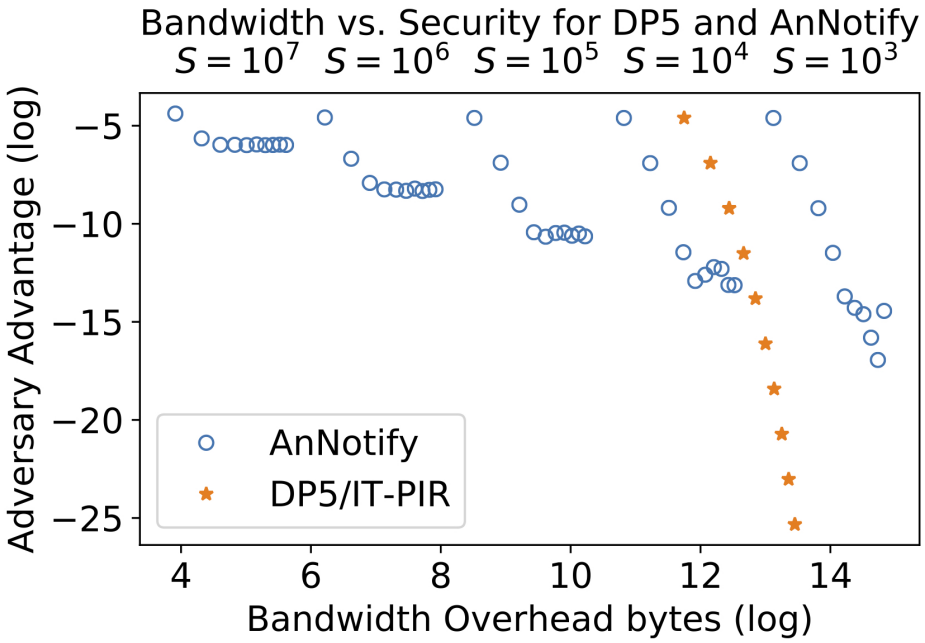


Figure 5.4: Security versus Bandwidth comparison for AnNotify and DP5/IT-PIR.

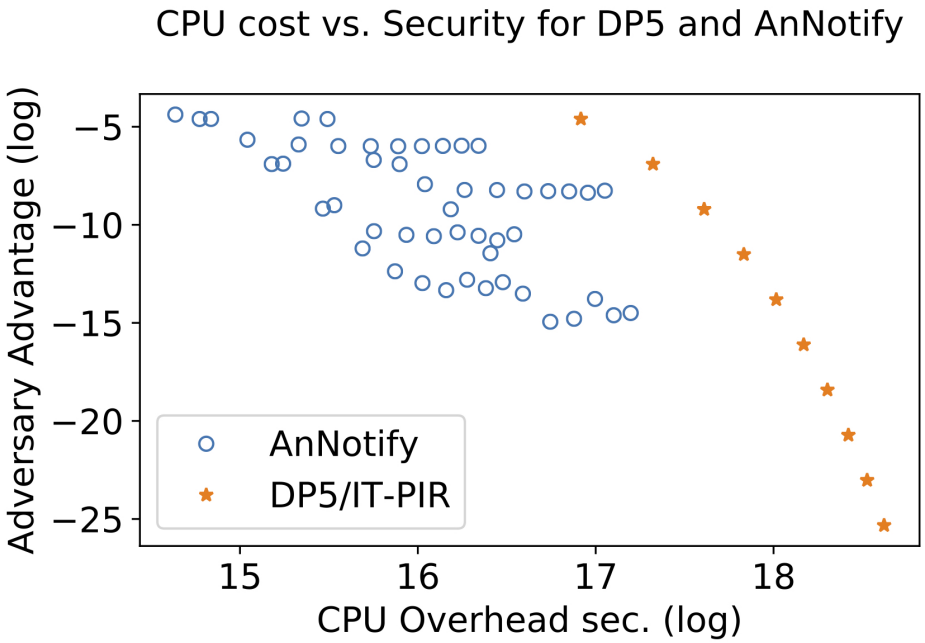


Figure 5.5: Security versus CPU cost comparison for AnNotify and DP5/IT-PIR.

per mix message using the Sphinx[9] packet format [177] for appropriate payload sizes and path lengths ℓ.

Figure 5.5 illustrates the total CPU costs for $u = 10^9$ queries and $f = 10\%$ for both DP5/IT-PIR and AnNotify. We vary $S \in \{10^3, \ldots, 10^8\}$ and $\ell \in \{2, \ldots, 11\}$.

[9]Using the Python `sphinxmix` package.

We observe that for equivalent security levels the CPU cost of mixing messages in AnNotify is always orders of magnitude (log scale x axis) lower than the equivalent CPU cost of processing IT-PIR queries in DP5.

5.7 AnNotify Extensions

AnNotify as a presence system. AnNotify can be used as a privacy-friendly presence system, to transmit a small amount of information from the publisher to the subscriber. A presence system allows users to indicate their online presence. For example, when a single user connects to the network the presence system informs which friends are online.

In this variant, each shard stores the received notifications as a list within each shard, instead of Bloom filter. Two users who would like to use AnNotify share a secret channel key ck. Alice wants to notify Bob of message m on this channel. To do so, she computes the value of a pseudo random function keyed with ck based on the current time stamp as $\mathsf{ID}^t = \mathsf{PRF}_{ck}(t)$ and the shard index $i = \mathsf{PRF}_{ck}(t) \bmod S$. She then encrypts the selected message with an Authenticated Encryption Scheme with Associated Data (AEAD) (such as AES-GCM) with a secret key ck to obtain the ciphertext $c^t = \mathsf{AEAD}_{ck}(\mathsf{ID}^t; \mathsf{m})$. In order to notify, Alice sends the tuple (ID^t, c) to the corresponding shard s_i based on ID^t. The server adds it to the stored values within that shard.

At the beginning of the next epoch, Bob queries the servers for shard s_i and downloads the full set of values stored within it. To check for the presence notifications, the subscriber searches in the list the tuple with the identifier $\mathsf{PRF}_{ck}(t)$, and checks and decrypts the attached ciphertext and tag using secret key ck in order to recover the notification message m.

We note that the shard compression achieved through Bloom filters is sacrificed in order to transmit the message m. However, the subscriber-publisher privacy of Alice and Bob are maintained. A rigorous proof of this would have to adapt the security definition based on the $\mathsf{IndNotExp}$ experiment to provide the adversary with the ID^t_A and ID^t_B identifiers for the target messages instead of the raw keys ck_A and ck_B to preserve the secrecy of the message. However, the rest of the proof and Security Theorem 1 do not need major modification to show query privacy and message secrecy.

We note this scheme is in effect a leaky PIR scheme [221], based on a secure anonymity infrastructure, and untrusted servers holding shards. Given our evaluation results, relating the adversary advantage to performance, such designs may be

a competitive alternative for other PIR related applications.

Broadcast AnNotify. The Security definitions and IndNotExp security game assumes that the adversary knows the notification key used by a target subscriber. Yet, they are still unable to determine whether they seek a specific notification. As a result, AnNotify can be extended to support broadcast notifications to a group, without difficulties.

In a broadcast scheme, the notifier distributes the secret notification key amongst a group of subscribers. Access control is required when publishing a notification to ensure it is genuine. This may be achieved using any authentication or non-repudiation scheme, since notifiers are not anonymous. All subscribers in the group share that key, and query each epoch on the basis of it.

Due to the security guarantees of Security Theorem 1, even if one of the subscribers in the group is corrupt – and shares the key with the adversary – they are not able to break subscriber privacy of another target user with greater advantage than the one-on-one AnNotify design.

5.8 Applications

Notification-only Applications. The first application is a privacy-preserving version of event-notification services, such as the popular Yo application [222]. Yo and similar applications allow one user to send a content-free notification to peer(s). In Yo, the receiving applications notify the user by transmitting the word "Yo", in text and audio. Such event notification services can be used for social purposes, as well as to provide simple information about events, e.g., Yo was used to warn Israeli citizens of missile strikes [223].

As each message is only a single bit, applying Bloom filter is ideal for this kind of communication. The Anonymous Yo server will maintain a Bloom filter, and an anonymous Yo message will be sent by turning on a few bits according to the shared keys. The client side application will periodically retrieve the Bloom filter and will prompt Yo from another client, if this client turned on the relevant bits.

The second application is *Anonymous Presence Services*. The goal of anonymous presence services is to allow users to indicate their 'presence', i.e., availability for online communication to their peers. It is one of the functionalities usually provided by social networks such as Skype and Facebook. A privacy-preserving presence protocol, providing presence indications to users while hiding their relationships, was presented in [189]. Their solution relies on expensive cryptography and

is rather complex to implement, whereas AnNotify provides an easier-to-implement and more efficient solution.

The third application is privacy-preserving *blacklists*, e.g., of phishing domain names. The goal is to allow a relying party, e.g., a browser or email server, to check if a given domain name (or other identifier) is 'blacklisted', without exposing the identity of the domain being queried. In particular, all major browsers use some 'safe browsing' blacklist to protect users from phishing and malware websites. Google Safe Browsing (GSB) alone accounts for a billion users to date [224]. To protect users privacy, clients do not lookup the suspect URL or domain-name, instead the query is for a cryptographic hash of the domain-name or URL. However, as already observed [225], providers can still identify the query. AnNotify provides an alternative which strongly protects privacy, and with comparable overhead. We note that Bloom filters are already widely used to improve efficiency of blacklists, e.g., see [226, 227].

In all applications, AnNotify allows preserving the privacy of users, by hiding the relationships between users and the notifications they receive. The use of AnNotify is easy, and has insignificant performance overhead in addition to the use of anonymous channels. However, notice that AnNotify exposes the *total* number of clients currently connected to the system. We believe this is not a concern in many applications. Indeed, many services publish an estimate of the number of online clients, e.g., see Tor metrics [228].

Privacy-Preserving Caching and Storage Services. A classical use for Bloom filters, is to improve the efficiency of caching and storage mechanisms, by allowing efficient detection when cached items were updated (or not). In particular, Bloom filters were used to improve the efficiency of web-caches [219, 229].

AnNotify can similarly improve the efficiency of caching and storage mechanisms, while also protecting privacy. This is especially important for privacy-preserving storage mechanisms such as Oblivious RAM [210, 211] and PIR [139], where each access involves significant overhead, hence avoiding unnecessary requests has a large impact on performance.

Due to its high efficiency, AnNotify can also be used to improve the privacy of web and DNS caches. In particular, web-users may use AnNotify to improve the efficiency of anonymous-browsing mechanisms such as Tor [228] and the use of AnNotify seems to offer significant performance improvements compared to existing proposals for protecting privacy of DNS users, see [212, 213, 214].

5.9 Conclusions

In this chapter, we have described AnNotify, design of an efficient and scalable private notification system, as an alternative for previous approaches like DP5 [189] that struggles to scale past 1 million users. AnNotify benefits from a mass of users: its key security parameters depend on the number of shards and anonymity set size of the underlying anonymity system. These may be tuned to provide meaningful privacy protection despite some leakage.

AnNotify lowers the quality of protection to achieve scalability but does so in a controlled and well-understood manner: the concrete security theorems presented indicate the advantage of the adversary. The tighter bounds and empirical estimates of leakage under repeated queries provide even stronger evidence that AnNotify can provide strong protection. This is particularly relevant for large-scale deployments and applications requiring notifications, that today benefit from no protection at all.

We showed that PIR schemes inspired by the AnNotify design and anonymous channels may be more competitive in terms of performance than those proposed so far, despite leakage and required large anonymity set. Pursuing this research direction would allow the wider deployment of private querying in general.

Chapter 6

Conclusions and Future work

The main goal of this thesis was to investigate and analyse how we can design and build real-world systems supporting anonymous online communication using mix networks. Working in this direction, we proposed a novel design of a mix network (Chapter 3), which offers better security guarantees than any existing solution for anonymous communications, yet without the very high-latencies traditionally associated with mix systems. Loopix is the first mix network design which combines strong anonymity, scalability, and performance, and can accommodate various real-time communications. Hence, it is suitable for many daily applications, including emails, instant messaging, cryptocurrency transactions etc.

Loopix resists both the powerful and sophisticated adversaries capable of observing all communications as well as those performing active attacks, such as trickling and flooding. The combination of continuous-time mixes and tuneable cover traffic allows determining the tradeoff between the expected end-to-end latency and anonymity, hence Loopix can support applications with various latency and bandwidth constraints. Moreover, the memoryless property of the exponential distribution used to sample the delays results in larger anonymity set in Loopix, comparing to batch-and-reorder based mixes, because the anonymity set is asymptotically infinite. Therefore, an increasing number of system users not only contributes to better privacy, as the anonymity set grows, but also makes the system faster, as less additional exponential delay needs to be added to anonymize the traffic, and the volume of cover traffic can be tunned down. This is a property that was not offered by any of the previous designs.

Furthermore, the way Loopix generates cover traffic conceals the users' communication patterns. Hence, the adversary cannot infer when a client is actively communicating or is just sending and receiving dummy packets. Hence, Loopix protects against statistical disclosure attacks, thus even long-term conversations

cannot be exposed (previous mix network designs could not offer such protection).

While most of the existing systems' designs offering strong anonymity scale vertically, Loopix scales horizontally, hence its overall capacity increases by adding more infrastructure nodes. This is crucial since anonymous communication systems need to provide scalability in order to enable widespread adoption.

While Loopix offers strong security against passive attacks, an adversary can exploit the open nature of such systems and run a large subset of nodes, which may appear to run the protocol but in reality, are under full adversarial control. Such malicious nodes might attempt to drop or delay processed packets to observe a changed behaviour in the network and in result compromise users' anonymity. Therefore, in our thesis (Chapter 4) we further explore the idea of client-generated loop messages (introduced in Chapter 3) to study how to detect and penalize malicious nodes performing active attacks. Although continuous loop cover messages have several useful applications: (1) allow the clients to take anonymously measurements of the network state, (2) detect $(n-1)$ attacks against clients or mixes, and (3) generate bi-directional cover traffic, the loop messages on their own do not allow detecting which of the infrastructure nodes misbehaved. Hence, sporadic active attacks performed by the malicious network, nodes are difficult to detect and prove. Therefore, we present the first systematic analysis using quantitative and composable measure of security against dropping attacks. Along with our analysis, we propose an efficient and scalable Miranda mechanism which allows the clients to check the quality of service offered by the mix nodes and detect when malicious nodes perform drop packets and gather evidence of their misbehaviour, which results in the exclusion of the dishonest mixes from the network. Our design does not require any computationally expensive primitives, a considerable additional network bandwidth or any disclosure of sensitive information about the processed traffic. In addition to the Miranda mechanism, we also study various community detection techniques which can be further explored to strengthen Miranda.

In the second part of this thesis, we focused on studying how mix networks can be further used in building privacy-enhancing technologies. We proposed AnNotify, a system for private notifications, which scales to millions of users at a low bandwidth and performance cost, hence offers much better efficiency than other approaches, like for example DP5 or traditional PIR. AnNotify's performance evaluation shows that it is indeed very efficient, and extremely cheap to operate in the real world, hence can be deployed within other systems in which private notifications are crucial. Moreover, AnNotify is not limited to just notifications, but can

also be deployed as a privacy-preserving presence system, or extended to support broadcast notifications to a group.

Using AnNotify as privacy-preserving storage mechanisms and private querying alternative to Oblivious RAM or PIR is an interesting research direction. Similarly, due to its high-efficiency AnNotify might also be used in the future to improve the privacy of web and DNS caches, for example, in Tor.

6.1 Limitations and Future work

The Loopix design raises several interesting research questions, which were not covered in the original Loopix design. During the work on the Katzenpost[1] project we already tackled several of those research questions including a mechanism for reliable message delivery based on Single-Use Reply Blocks (SURBs), and message fragmentation and retransmission protocol. Furthermore, we proposed how Loopix can be used to add network-level privacy for cryptocurrencies like Zcash in [1], which was later also added to Katzenpost [2]. However, there are still open research questions remaining. The Loopix discrete simulator (presented in Section 3.6 in Chapter 3) allows empirically measuring the anonymity and performance of the network, given different configuration parameters of the topology, the number of users or latency-overhead demands. An interesting question is whether it is possible to find dependency between those parameters in order to develop a methodology to optimally tune the trade-offs between parameters of the mix network under real-world, non-ideal conditions, to obtain the best possible anonymity and performance, and whether such parameters could be further adjusted dynamically in a privacy-preserving manner.

Furthermore, Loopix assumes a fixed sending rate for all users (λ_P). While good for privacy, it becomes restrictive if we want to adopt Loopix for various applications. Thus, a potential solution is to have a set of several fixed rates (e.g., fast, medium, slow), which the users might pick depending on the application they would like to use. Thus, the users' distinctive communication patterns are still hidden (among all the other users using the same rate) but at the same time, Loopix becomes a generic communication infrastructure that can support a broad range of applications and services, combining all the traffic into one large anonymity set. An interesting question is what the fast, medium and slow rates should be, in order to support a wide range of applications, but at the same time still protect users from long term disclosure attacks. However, it is worth to note, that the independent

[1]Reminder: Katzenpost is the open-source Loopix based mix network implementation funded by EU Panoramix project https://github.com/katzenpost/docs

streams of loop and drop cover traffic allow the users to control their own privacy, i.e., an individual can increase the volume of cover traffic arbitrarily to obfuscate their communication pattern. On the other hand, if an individual user decides to turn off the loop and drop cover traffic they risk exposing their communication patterns, but they do not harm the privacy of other users who use the loop and drop cover traffic.

Another research question is the idea of an incentivized mix network. Large scale anonymous communication networks, like Tor or earlier mix network deployments Mixmaster and Mixminion, rely on a voluntaristic model, which leads to poor scalability and performance. Strong anonymity against traffic analysis is based on distributed trust, where the infrastructure consists of multiple independently controlled relays that form a united network. Also, splitting the bandwidth among many mixes allows avoiding performance bottlenecks. However, the cost of maintaining these nodes for a long period is significant, considering the high volume of bandwidth they have to handle and currently the only incentive for someone to do it is the feeling of providing community service. Since the quality of the anonymity service provided depends on the number of mixes and their incentive to operate honestly, there is a strong need for giving economic incentives to people in order to maintain such nodes. As an example, VPNs or cryptocurrencies (including Bitcoin, Zcash, Monero, and cryptocurrency tumblers) show that users are willing to pay for their privacy. On the other hand, the reward-based model used in cryptocurrencies incentivizes users to join the network and operate efficiently and honestly, while the consensus algorithms like proof of stake make Sybil attacks impractical. This research question inspired the Nym Technologies company to create the Nym network, a permissionless and incentivised mix network infrastructure.

As mentioned in Chapter 3, the providers act as storage points for clients who are offline or unreachable and mediate the users' access to the mix network, acting as the entry-exit gateway points. Each provider has a long-term relationship with its users and may authenticate them, potentially bill them, or discontinue their access to the network. This, as result, allows the providers to prevent congestions and DoS attacks. We explore this idea further at Nym, by adding Coconut credentials [230], which allow to bill the clients, and prevent abusive use of the network.

While the Miranda mechanism allows to effectively detect and exclude malicious nodes, there still remain practical open problems concerning the exposure of malicious nodes. The most significant simplifying assumption in Miranda are: (1) fixed set of mixes, (2) majority of benign mixes, and (3) reliable communication and processing. Such assumptions are very limiting in terms of practical deployment, and therefore should be considered as open research questions. Future

work should try to avoid these assumptions while maintaining a thorough security analysis and properties as done in Miranda, or identify any inherent trade-offs. In practice, communication and processing failures will happen - in particular, as a result of intentional DoS attacks. We believe that the future work may deal with this significant challenge by both *minimizing failures*, by designing robust underlying mechanisms such as a highly-resilient transport layer; and *refined assumptions and analysis*, e.g., considering incentives and game-theory analysis, to ensure that the system is robust to 'reasonable' levels of failures. These issues are significant challenges for future research, essential towards the implementation of Miranda in practical systems.

Moreover, a practical mixnet must allow a dynamic set of mixes, for both scalability and churn - mixes joining and leaving over time. The system needs the ability to naturally grow to allow scalability, while the infrastructure nodes in practice are not guaranteed to be available and might naturally join/leave the network constantly. On the other hand, if the adversary can simply retire penalized malicious nodes and replace them with new nodes that have an untarnished reputation, then there is no real gain in even trying to penalize or expose the adversary, and it becomes hard to argue why we can even assume most mixes are benign. Therefore, further research must develop a reasonable model to allow nodes to join (or re-join) without allowing the adversary to gain the majority by adding many mixes, as in Sybil attacks, and to retain the impact of removing corrupt mixes. The mentioned earlier incentives model might offer a solution for some of those questions.

Overall, the research presented in this thesis introduces novel designs and analysis towards better anonymous communications, which set in motion the development of open-source software and the deployment of a privacy infrastructure which is further continued by Nym Technologies company. While the Loopix mix network was deployed by Nym Technologies, the ideas proposed in the Miranda mechanism (i.e., using loop packets to verify honest behaviours of the nodes, combined with the packet receipts) inspired the proof-of-mixing developed by Nym. As the Head of Research at Nym Technologies, I have the opportunity to further extend this research and be part of the large-scale development of mix network infrastructure.

Bibliography

[1] George Kappos and Ania M. Piotrowska. *"Extending the Anonymity of Zcash"*. CoRR, abs/1902.07337, 2019. http://arxiv.org/abs/1902.07337.

[2] Using Zcash with Katzenpost, 2019. https://katzenpost.mixnetworks.org/zcash.html.

[3] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. *"Denial of service or denial of security?"* In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 92–102. ACM, 2007.

[4] Carole Cadwalladr and Emma Graham-Harrison. *"Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach"*. The Guardian. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election.

[5] Issie Lapowsky. *"Cambridge Analytica Took 50M Facebook Users' Data—And Both Companies Owe Answers"*. . https://www.wired.com/story/cambridge-analytica-50m-facebook-users-data/.

[6] Issie Lapowsky. *"Facebook Gave a Russian Internet Giant a Special Data Extension"*. . https://www.wired.com/story/facebook-gave-russian-internet-giant-special-data-extension/.

[7] Glenn Greenwald and Ewen MacAskill. *"NSA Prism program taps in to user data of Apple, Google and others"*. The Guardian. https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[8] *"NSA slides explain the PRISM data-collection program"*. The Washington Post. https://www.washingtonpost.com/wp-srv/special/politics/prism-collection-documents/.

[9] Glenn Greenwald. *"NSA collecting phone records of millions of Verizon customers daily"*. The Guardian. https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order.

[10] Trailblazer Project. https://en.wikipedia.org/wiki/Trailblazer_Project.

[11] Protect America Act. https://www.justice.gov/archive/ll/index.html.

[12] Kadhim Shubber. *"A simple guide to GCHQ's internet surveillance programme Tempora"*. Wired. https://www.wired.co.uk/article/gchq-tempora-101.

[13] Harry Davies. *"Ted Cruz using firm that harvested data on millions of unwitting Facebook users"*. The Guardian. https://www.theguardian.com/us-news/2015/dec/11/senator-ted-cruz-president-campaign-facebook-user-data.

[14] Open PGP. https://www.openpgp.org/.

[15] TLS Documentation. https://tools.ietf.org/html/rfc5246.

[16] WhatsApp. https://www.whatsapp.com/.

[17] Signal. https://signal.org/.

[18] Ian Goldberg and OTR Development Team. Off-the-record communication. https://otr.cypherpunks.ca/.

[19] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. *"Off-the-record communication, or, why not to use PGP"*. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84. ACM, 2004.

[20] Chris Conley. *"Metadata: Piecing together a privacy solution"*. SSNR, 2014. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2573962.

[21] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. *"Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks"*. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 3–18. IEEE Computer Society, 2011.

[22] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian. *"Timing Analysis of Keystrokes and Timing Attacks on SSH"*. In Dan S. Wallach, editor, *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*, 2001.

[23] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. *"Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail"*. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 332–346. IEEE Computer Society, 2012.

[24] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan R. Mayer, Arvind Narayanan, and Edward W. Felten. *"Cookies That Give You Away: The Surveillance Implications of Web Tracking"*. In Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 289–299. ACM, 2015.

[25] Zcash. https://z.cash.

[26] Evan Duffield and Daniel Diaz. *"Dash: A PrivacyCentric CryptoCurrency"*. 2015. https://github.com/dashpay/dash/wiki/Whitepaper.

[27] Dash. https://www.dash.org.

[28] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. *"Zerocash: Decentralized Anonymous Payments from Bitcoin"*. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014.

[29] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *"Zcash protocol specification"*. Technical report, Zerocoin Electric Coin Company, 2016.

[30] Monero. https://getmonero.org.

[31] Helger Lipmaa. *"Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes"*. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, pages 41–60. Springer, 2013.

[32] Jens Groth. *"On the Size of Pairing-Based Non-interactive Arguments"*. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 305–326. Springer, 2016.

[33] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. *"1-out-of-N Signatures from a Variety of Keys"*. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, pages 415–432. Springer, 2002.

[34] Shifeng Sun, Man Ho Au, Joseph K. Liu, and Tsz Hon Yuen. *"RingCT 2.0: A Compact Accumulator-Based (Linkable Ring Signature) Protocol for Blockchain Cryptocurrency Monero"*. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, pages 456–474. Springer, 2017.

[35] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. *"Eclipse*

Attacks on Bitcoin's Peer-to-Peer Network". In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, Washington, D.C., USA, August 12-14, 2015*, pages 129–144, 2015.

[36] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller, and Bobby Bhattacharjee. *"TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions"*. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, pages 550–566. Springer, 2019.

[37] Lee Ferran. *"Ex-NSA Chief:'We Kill People Based on Metadata'"*. ABC News Blogs, 2014. https://abcnews.go.com/blogs/headlines/2014/05/ex-nsa-chief-we-kill-people-based-on-metadata.

[38] Bruce Schneier. *"NSA doesn't need to spy on your calls to learn your secrets"*. Wired, 2015. https://www.wired.com/2015/03/data-and-goliath-nsa-metadata-spying-your-secrets/.

[39] Microsoft. Virtual Private Networking: An Overview. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/bb742566(v=technet.10).

[40] Andrew G Mason. *Cisco secure virtual private networks*. Cisco Press, 2001.

[41] Lance Cottrell. Anonymizer. https://www.anonymizer.com/.

[42] *"The few behind many: hidden VPN owners unveiled"*. https://vpnpro.com/wp-content/uploads/Infographic-VPNpro-97-VPN-products-run-by-just-23-companies.pdf.

[43] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. *"Tor: The Second-Generation Onion Router"*. In Matt Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320, 2004.

[44] Paul F. Syverson, Gene Tsudik, Michael G. Reed, and Carl E. Landwehr. *"Towards an Analysis of Onion Routing Security"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 96–114. Springer, 2000.

[45] Steven J. Murdoch. *"Hot or not: revealing hidden services by their clock skew"*. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 27–36. ACM, 2006.

[46] Lasse Øverlier and Paul F. Syverson. *"Locating Hidden Servers"*. In *2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*, pages 100–114. IEEE Computer Society, 2006.

[47] Steven J. Murdoch and George Danezis. *"Low-Cost Traffic Analysis of Tor"*. In *2005 IEEE Symposium on Security and Privacy (S&P 2005), 8-11 May 2005, Oakland, CA, USA*, pages 183–195. IEEE Computer Society, 2005.

[48] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. *"Touching from a distance: website fingerprinting attacks and defenses"*. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 605–616. ACM, 2012.

[49] David Chaum. *"Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms"*. In Dimitris Gritzalis, editor, *Secure Electronic Voting*, volume 7 of Advances in Information Security, pages 211–219. Springer, 2003.

[50] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. *"Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two"*. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 108–126. IEEE Computer Society, 2018.

[51] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. *"The Loopix Anonymity System"*. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, Vancouver, BC, Canada, August 16-18, 2017*, pages 1199–1216, 2017.

[52] Hemi Leibowitz, Ania M. Piotrowska, George Danezis, and Amir Herzberg. *"No Right to Remain Silent: Isolating Malicious Mixes"*. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, Santa Clara, CA, USA, August 14-16, 2019*, pages 1841–1858, 2019.

[53] Ania M. Piotrowska, Jamie Hayes, Nethanel Gelernter, George Danezis, and Amir Herzberg. *"AnNotify: A Private Notification Service"*. In Bhavani M. Thuraisingham and Adam J. Lee, editors, *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society, Dallas, TX, USA, October 30 - November 3, 2017*, pages 5–15. ACM, 2017.

[54] Claude E. Shannon. *"A mathematical theory of communication"*. Mobile Computing and Communications Review, 5(1):3–55, 2001. URL https://doi.org/10.1145/584091.584093.

[55] Cynthia Dwork and Aaron Roth. *"The Algorithmic Foundations of Differential Privacy"*. Foundations and Trends in Theoretical Computer Science, 9 (3-4):211–407, 2014. URL https://doi.org/10.1561/0400000042.

[56] Burton H. Bloom. *"Space/Time Trade-offs in Hash Coding with Allowable Errors"*. Commun. ACM, 13(7):422–426, 1970. URL https://doi.org/10.1145/362686.362692.

[57] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[58] Andreas Pfitzmann and Marit Köhntopp. *"Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 1–9. Springer, 2000.

[59] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. *"The Disadvantages of Free MIX Routes and how to Overcome Them"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 30–45. Springer, 2000.

[60] Andrei Serjantov and George Danezis. *"Towards an Information Theoretic Metric for Anonymity"*. In Roger Dingledine and Paul F. Syverson, editors, *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, pages 41–53. Springer, 2002.

[61] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. *"Towards Measuring Anonymity"*. In Roger Dingledine and Paul F. Syverson, editors, *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, pages 54–68. Springer, 2002.

[62] George Danezis. *"Statistical Disclosure Attacks"*. In Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis K. Katsikas, editors, *Security and Privacy in the Age of Uncertainty, IFIP TC11 18th International Conference on Information Security (SEC2003), May 26-28, 2003, Athens, Greece*, pages 421–426. Kluwer, 2003.

[63] George Danezis and Andrei Serjantov. *"Statistical Disclosure or Intersection Attacks on Anonymity Systems"*. In Jessica J. Fridrich, editor, *Information Hiding, 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers*, pages 293–308. Springer, 2004.

[64] Paul F. Syverson, Michael G. Reed, and David M. Goldschlag. *"Private Web Browsing"*. Journal of Computer Security, 5(3):237–248, 1997. URL https://doi.org/10.3233/JCS-1997-5305.

[65] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. *"Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier"*. In Radu Sion and Dawn Song, editors, *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009, Chicago, IL, USA, November 13, 2009*, pages 31–42. ACM, 2009.

[66] Andrew Hintz. *"Fingerprinting Websites Using Traffic Analysis"*. In Roger Dingledine and Paul F. Syverson, editors, *Privacy Enhancing Technologies, Second International Workshop, PETS 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, pages 171–178. Springer, 2002.

[67] Jean-François Raymond. *"Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 10–29. Springer, 2000.

[68] Andrei Serjantov, Roger Dingledine, and Paul F. Syverson. *"From a Trickle to a Flood: Active Attacks on Several Mix Types"*. In Fabien A. P. Petitcolas, editor, *Information Hiding, 5th International Workshop, IH 2002, Noordwijkerhout, The Netherlands, October 7-9, 2002, Revised Papers*, pages 36–52. Springer, 2002.

[69] Luke O'Connor. *"On Blending Attacks for Mixes with Memory"*. In Mauro Barni, Jordi Herrera-Joancomartí, Stefan Katzenbeisser, and Fernando Pérez-González, editors, *Information Hiding, 7th International Workshop, IH 2005, Barcelona, Spain, June 6-8, 2005, Revised Selected Papers*, pages 39–52. Springer, 2005.

[70] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. *"An Analysis of the Degradation of Anonymous Protocols"*. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2002, San Diego, California, USA*, pages 1–12. The Internet Society, 2002.

[71] Birgit Pfitzmann and Andreas Pfitzmann. *"How to Break the Direct RSA-Implementation of Mixes"*. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, pages 373–381. Springer, 1989.

[72] Ceki Gülcü and Gene Tsudik. *"Mixing Email with Babel"*. In James T. Ellis, B. Clifford Neuman, and David M. Balenson, editors, *1996 Symposium on Network and Distributed System Security, NDSS '96, San Diego, CA, USA, February 22-23, 1996*, pages 2–16. IEEE Computer Society, 1996.

[73] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster anonymous remailer, 2004. http://mixmaster.sourceforge.net/.

[74] George Danezis, Roger Dingledine, and Nick Mathewson. *"Mixminion: Design of a Type III Anonymous Remailer Protocol"*. In *2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA*, pages 2–15. IEEE Computer Society, 2003.

[75] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. *"ISDN-MIXes: Untraceable Communication with Small Bandwidth Overhead"*. In Wolfgang Effelsberg, Hans Werner Meuer, and Günter Müller, editors, *Kommunikation in Verteilten Systemen, Grundlagen, Anwendungen, Betrieb, GI/ITG-Fachtagung, Mannheim, 20.-22. Februar 1991, Proceedings*, pages 451–463. Springer, 1991.

[76] Anja Jerichow, Jan Müller, Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. *"Real-time mixes: a bandwidth-efficient anonymity protocol"*. IEEE Journal on Selected Areas in Communications, 16(4):495–509, 1998. URL https://doi.org/10.1109/49.668973.

[77] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. *"Web MIXes: A System for Anonymous and Unobservable Internet Access"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 115–129. Springer, 2000.

[78] Roger Dingledine, Andrei Serjantov, and Paul F. Syverson. *"Blending Different Latency Traffic with Alpha-mixing"*. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies, 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, pages 245–257. Springer, 2006.

[79] Markus Jakobsson. *"Flash Mixing"*. In Brian A. Coan and Jennifer L. Welch, editors, *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, '99Atlanta, Georgia, USA, May 3-6, 1999*, pages 83–89. ACM, 1999.

[80] Masashi Mitomo and Kaoru Kurosawa. *"Attack for Flash MIX"*. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 192–204. Springer, 2000.

[81] Andrei Serjantov and Richard E. Newman. *"On the Anonymity of Timed Pool Mixes"*. In Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis K. Katsikas, editors, *Security and Privacy in the*

Age of Uncertainty, IFIP TC11 18th International Conference on Information Security (SEC2003), May 26-28, 2003, Athens, Greece, pages 427–434. Kluwer, 2003.

[82] Dogan Kesdogan, Jan Egner, and Roland Büschkes. *"Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System"*. In David Aucsmith, editor, *Information Hiding, Second International Workshop, Portland, Oregon, USA, April 14-17, 1998, Proceedings*, pages 83–98. Springer, 1998.

[83] Masayuki Abe. *"Mix-Networks on Permutation Networks"*. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings*, pages 258–273. Springer, 1999.

[84] Stephanie Bayer and Jens Groth. *"Efficient Zero-Knowledge Argument for Correctness of a Shuffle"*. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 263–280. Springer, 2012.

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

[86] Jun Furukawa and Kazue Sako. *"An Efficient Scheme for Proving a Shuffle"*. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 368–387. Springer, 2001.

[87] C. Andrew Neff. *"A verifiable secret shuffle and its application to e-voting"*. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 116–125. ACM, 2001.

[88] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. *"Universal Re-encryption for Mixnets"*. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, pages 163–178. Springer, 2004.

[89] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. *"Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking"*. In Dan Boneh, editor, *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002*, pages 339–353, 2002.

[90] Shahram Khazaei and Douglas Wikström. *"Randomized Partial Checking Revisited"*. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 115–128. Springer, 2013.

[91] David Chaum. *"The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability"*. Journal of Cryptology, 1(1):65–75, 1988. URL https://doi.org/10.1007/BF00206326.

[92] Michael Waidner and Birgit Pfitzmann. *"The Dining Cryptographers in the Disco - Underconditional Sender and Recipient Untraceability with Computationally Secure Serviceability "*. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, page 690. Springer, 1989.

[93] Philippe Golle and Ari Juels. *"Dining Cryptographers Revisited"*. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 456–473. Springer, 2004.

[94] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. *"Herbivore: A scalable and efficient protocol for anonymous communication"*. Technical report, Cornell University, 2003. https://hdl.handle.net/1813/5606.

[95] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. *"How much anonymity does network latency leak?"* ACM Trans. Inf. Syst. Secur., 2010.

[96] Juan A. Elices and Fernando Pérez-González. *"Fingerprinting a flow of messages to an anonymous server"*. In *2012 IEEE International Workshop on Information Forensics and Security, WIFS 2012, Costa Adeje, Tenerife, Spain, December 2-5, 2012*, pages 97–102. IEEE, 2012.

[97] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. *"A Critical Evaluation of Website Fingerprinting Attacks"*. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 263–274. ACM, 2014.

[98] Jamie Hayes and George Danezis. *"k-fingerprinting: A Robust Scalable Website Fingerprinting Technique"*. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, Austin, TX, USA, August 10-12, 2016*, pages 1187–1203, 2016.

[99] Juan A. Elices, Fernando Pérez-González, and Carmela Troncoso. *"Finger-*

printing Tor's hidden service log files using a timing channel". In *2011 IEEE International Workshop on Information Forensics and Security, WIFS 2011, Iguacu Falls, Brazil, November 29 - December 2, 2011*, pages 1–6. IEEE Computer Society, 2011.

[100] Michael K. Reiter and Aviel D. Rubin. *"Crowds: Anonymity for Web Transactions"*. ACM Trans. Inf. Syst. Secur., 1998.

[101] Michael J. Freedman and Robert Tappan Morris. *"Tarzan: a peer-to-peer anonymizing network layer"*. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, 2002.

[102] Marc Rennhard and Bernhard Plattner. *"Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection"*. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, 2002.

[103] Prateek Mittal, Matthew K. Wright, and Nikita Borisov. *"Pisces: Anonymous Communication Using Social Networks"*. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013.

[104] Arjun Nambiar and Matthew K. Wright. *"Salsa: a structured approach to large-scale anonymity"*. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, 2006.

[105] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S. Wallach. *"AP3: cooperative, decentralized anonymous communication"*. In Yolande Berbers and Miguel Castro, editors, *Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22, 2004*, 2004.

[106] Andriy Panchenko, Stefan Richter, and Arne Rache. *"NISAN: network information service for anonymization networks"*. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, 2009.

[107] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. *"Scalable onion routing with torsk"*. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, Chicago, Illinois, USA, November 9-13, 2009*, 2009.

[108] Hari Balakrishnan, M. Frans Kaashoek, David R. Karger, Robert Tappan

Morris, and Ion Stoica. *"Looking up data in P2P systems"*. Commun. ACM, 2003.

[109] Matthew K Wright, Micah Adler, Brian Neil Levine, and Clay Shields. *"The predecessor attack: An analysis of a threat to anonymous communications systems"*. ACM Transactions on Information and System Security (TISSEC), 2004.

[110] George Danezis and Richard Clayton. *"Route Fingerprinting in Anonymous Communications"*. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P 2006), 2-4 October 2006, Cambridge, United Kingdom*, 2006.

[111] Parisa Tabriz and Nikita Borisov. *"Breaking the Collusion Detection Mechanism of MorphMix"*. In George Danezis and Philippe Golle, editors, *Privacy Enhancing Technologies, 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, 2006.

[112] George Danezis, Claudia Díaz, Carmela Troncoso, and Ben Laurie. *"Drac: An Architecture for Anonymous Low-Volume Communications"*. In *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, 2010.

[113] Shishir Nagaraja. *"Anonymity in the Wild: Mixes on Unstructured Networks"*. In *Privacy Enhancing Technologies, 7th International Symposium, PET 2007 Ottawa, Canada, June 20-22, 2007, Revised Selected Papers*, 2007.

[114] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. *"Systematizing Decentralization and Privacy: Lessons from 15 Years of Research and Deployments"*. Proceedings of Privacy Enhancing Technologies, 2017.

[115] George Danezis and Paul F. Syverson. *"Bridging and Fingerprinting: Epistemic Attacks on Route Selection"*. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008, Leuven, Belgium, July 23-25, 2008, Proceedings*, 2008.

[116] Prateek Mittal and Nikita Borisov. *"Information Leaks in Structured Peer-to-Peer Anonymous Communication Systems"*. 2012.

[117] Qiyan Wang, Prateek Mittal, and Nikita Borisov. *"In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems"*. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer*

and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010, 2010.

[118] John Brooks. Ricochet. https://ricochet.im/.

[119] Jonathan Warren. *"Bitmessage: A peer-to-peer message authentication and delivery system"*. 2012. https://bitmessage.org/bitmessage.pdf.

[120] Bitmessage. https://github.com/Bitmessage.

[121] Rob Sherwood, Bobby Bhattacharjee, and Aravind Srinivasan. *"P5: A Protocol for Scalable Anonymous Communication"*. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 58–70. IEEE Computer Society, 2002.

[122] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. *"Freenet: A Distributed Anonymous Information Storage and Retrieval System"*. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings*, pages 46–66. Springer, 2000.

[123] Bassam Zantout, Ramzi Haraty, et al. *"I2P data communication system"*. In *Proceedings of ICN*, pages 401–409. Citeseer, 2011.

[124] The Kovri Project. https://gitlab.com/kovri-project/kovri/blob/master/README.md.

[125] Juan Pablo Timpanaro, Chrisment Isabelle, and Festor Olivier. *"Monitoring the I2P network"*. 2011.

[126] Christoph Egger, Johannes Schlumberger, Christopher Kruegel, and Giovanni Vigna. *"Practical Attacks against the I2P Network"*. In Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright, editors, *Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings*, pages 432–451. Springer, 2013.

[127] Adrian Crenshaw. *"Darknets and hidden servers: Identifying the true IP/network identity of I2P service hosts"*. Black Hat DC, 201(1):1–23, 2011. http://www.irongeek.com/i.php?page=security/darknets-i2p-identifying-hidden-servers.

[128] Naoum Naoumov and Keith W. Ross. *"Exploiting P2P systems for DDoS attacks"*. In Xiaohua Jia, editor, *Proceedings of the 1st International Conference on Scalable Information Systems, Infoscale 2006, Hong Kong, May 30-June 1, 2006*, 2006.

[129] Miguel Castro, Peter Druschel, Ayalvadi J. Ganesh, Antony I. T. Rowstron, and Dan S. Wallach. *"Secure Routing for Structured Peer-to-Peer Over-*

lay Networks". In *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*. USENIX Association, 2002.

[130] Lin Wang. *"Attacks against peer-to-peer networks and countermeasures"*. In *T-110.5290 Seminar on Network Security*, 2006.

[131] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross J. Anderson. *"Sybil-Resistant DHT Routing"*. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 305–318. Springer, 2005.

[132] John R. Douceur. *"The Sybil Attack"*. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, pages 251–260. Springer, 2002.

[133] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. *"Vuvuzela: scalable private messaging resistant to traffic analysis"*. In Ethan L. Miller and Steven Hand, editors, *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, pages 137–152. ACM, 2015.

[134] Cynthia Dwork. *"Differential Privacy"*. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, pages 1–12. Springer, 2006.

[135] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. *"Stadium: A Distributed Metadata-Private Messaging System"*. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 423–440. ACM, 2017.

[136] David Lazar, Yossi Gilad, and Nickolai Zeldovich. *"Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis"*. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 711–725, 2018.

[137] Nethanel Gelernter, Amir Herzberg, and Hemi Leibowitz. *"Two Cents for Strong Anonymity: The Anonymous Post-office Protocol"*. In Srdjan Capkun and Sherman S. M. Chow, editors, *Cryptology and Network Security - 16th International Conference, CANS 2017, Hong Kong, China, November 30 - December 2, 2017, Revised Selected Papers*, pages 390–412. Springer, 2017.

[138] Sebastian Angel and Srinath T. V. Setty. *"Unobservable Communication over Fully Untrusted Infrastructure"*. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 551–569, 2016.

[139] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. *"Private Information Retrieval"*. J. ACM, 45(6):965–981, 1998. URL https://doi.org/10.1145/293347.293350.

[140] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. *"MCMix: Anonymous Messaging via Secure Multiparty Computation"*. In Engin Kirda and Thomas Ristenpart, editors, *26th USENIX Security Symposium, Vancouver, BC, Canada, August 16-18, 2017*, pages 1217–1234, 2017.

[141] Oded Goldreich, Silvio Micali, and Avi Wigderson. *"How to play any mental game, or a completeness theorem for protocols with honest majority"*. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. ACM, 2019.

[142] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. *"cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations"*. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, pages 557–578. Springer, 2017.

[143] Herman Galteland, Stig Fr. Mjølsnes, and Ruxandra F. Olimid. *"Attacks on cMix - Some Small Overlooked Details"*. IACR Cryptology ePrint Archive, 2016:729, 2016. URL http://eprint.iacr.org/2016/729.

[144] Albert Kwon, David Lu, and Srinivas Devadas. *"XRD: Scalable Messaging System with Cryptographic Privacy"*. CoRR, abs/1901.04368, 2019. URL http://arxiv.org/abs/1901.04368.

[145] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. *"Dissent in Numbers: Making Strong Anonymity Scale"*. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 179–182, 2012.

[146] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. *"Riposte: An Anonymous Messaging System Handling Millions of Users"*. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 321–338. IEEE Computer Society, 2015.

[147] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. *"Riffle: An Efficient Communication System With Strong Anonymity"*. PoPETs, 2016(2): 115–134, 2016. URL https://doi.org/10.1515/popets-2016-0008.

[148] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou, and Thomas A. Berson. *"How to Explain Zero-Knowledge Protocols to Your Children"*. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 628–631. Springer, 1989.

[149] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[150] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. *"Atom: Scalable Anonymity Resistant to Traffic Analysis"*. CoRR, abs/1612.07841, 2016. URL http://arxiv.org/abs/1612.07841.

[151] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. *"The Byzantine Generals Problem"*. ACM Trans. Program. Lang. Syst., 4(3):382–401, 1982. URL http://doi.acm.org/10.1145/357172.357176.

[152] Stevens Le Blond, David R. Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. *"Towards efficient traffic-analysis resistant anonymity networks"*. In Dah Ming Chiu, Jia Wang, Paul Barford, and Srinivasan Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 303–314. ACM, 2013.

[153] Pierangela Samarati and Latanya Sweeney. *"Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression"*. Technical report, technical report, SRI International, 1998.

[154] Stevens Le Blond, David R. Choffnes, William Caldwell, Peter Druschel, and Nicholas Merritt. *"Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems"*. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pages 639–652. ACM, 2015.

[155] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Adrian Perrig, Akira Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. *"LAP: Lightweight Anonymity and Privacy"*. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 506–520. IEEE Computer Society, 2012.

[156] Jody Sankey and Matthew K. Wright. *"Dovetail: Stronger Anonymity in Next-Generation Internet Routing"*. In Emiliano De Cristofaro and Steven J. Murdoch, editors, *Privacy Enhancing Technologies - 14th International Symposium, PETS 2014, Amsterdam, The Netherlands, July 16-18, 2014. Proceedings*, pages 283–303. Springer, 2014.

[157] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. *"HORNET: High-speed Onion Routing at the Network Layer"*. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1441–1454. ACM, 2015.

[158] Chen Chen, Daniele Enrico Asoni, Adrian Perrig, David Barrera, George Danezis, and Carmela Troncoso. *"TARANET: Traffic-Analysis Resistant Anonymity at the Network Layer"*. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*, pages 137–152. IEEE, 2018.

[159] Loki Network. https://loki.network/.

[160] Orchid: A decentralized network routing market, 2019. https://www.orchid.com/assets/whitepaper/whitepaper.pdf.

[161] Vpn0: A privacy-preserving distributed virtual private network, 2019. https://brave.com/vpn0-a-privacy-preserving-distributed-virtual-private-network/.

[162] Sentinel. https://sentinel.co/.

[163] Mysterium network project. https://mysterium.network/whitepaper.pdf.

[164] Roger Dingledine, Vitaly Shmatikov, and Paul F. Syverson. *"Synchronous Batching: From Cascades to Free Routes"*. In David M. Martin Jr. and Andrei Serjantov, editors, *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, pages 186–206. Springer, 2004.

[165] George Danezis. *"Mix-Networks with Restricted Routes"*. In Roger Dingledine, editor, *Privacy Enhancing Technologies, Third International Workshop, PET 2003, Dresden, Germany, March 26-28, 2003, Revised Papers*, pages 1–17. Springer, 2003.

[166] Claudia Díaz, George Danezis, Christian Grothoff, Andreas Pfitzmann, and Paul F. Syverson. *"Panel Discussion - Mix Cascades Versus Peer-to-Peer: Is One Concept Superior?"* In David M. Martin Jr. and Andrei Serjantov, editors, *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, 2004.

[167] Claudia Díaz, Steven J. Murdoch, and Carmela Troncoso. *"Impact of Network Topology on Anonymity and Overhead in Low-Latency Anonymity Networks"*. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, 2010.

[168] Joan Feigenbaum, Aaron Johnson, and Paul F. Syverson. *"Preventing Active Timing Attacks in Low-Latency Anonymous Communication"*. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, 2010.

[169] Ryan Henry and Ian Goldberg. *"Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting"*. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*, pages 71–82. ACM, 2013.

[170] Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. *"PAR: Payment for Anonymous Routing"*. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008, Leuven, Belgium, July 23-25, 2008, Proceedings*, pages 219–236. Springer, 2008.

[171] David Lazar and Nickolai Zeldovich. *"Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata"*. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 571–586, 2016.

[172] Hitesh Ballani, Paul Francis, and Xinyang Zhang. *"A study of prefix hijacking and interception in the internet"*. In Jun Murai and Kenjiro Cho, editors, *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, pages 265–276. ACM, 2007.

[173] Yossi Gilad and Amir Herzberg. *"Spying in the Dark: TCP and Tor Traffic Analysis"*. In Simone Fischer-Hübner and Matthew K. Wright, editors, *Privacy Enhancing Technologies - 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings*, pages 100–119. Springer, 2012.

[174] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. *"AnoA: A Framework for Analyzing Anonymous Communication Protocols"*. In *2013 IEEE 26th Computer Security Founda-*

tions Symposium, New Orleans, LA, USA, June 26-28, 2013, pages 163–178. IEEE Computer Society, 2013.

[175] Matthew K. Wright, Micah Adler, Brian Neil Levine, and Clay Shields. *"Passive-Logging Attacks Against Anonymous Communications Systems"*. ACM Trans. Inf. Syst. Secur., 2008.

[176] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. *"Users get routed: traffic correlation on tor by realistic adversaries"*. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 337–348. ACM, 2013.

[177] George Danezis and Ian Goldberg. *"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*, pages 269–282. IEEE Computer Society, 2009.

[178] Ross J. Anderson and Eli Biham. *"Two Practical and Provably Secure Block Ciphers: BEARS and LION"*. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, pages 113–120. Springer, 1996.

[179] George Danezis. *"The Traffic Analysis of Continuous-Time Mixes"*. In David M. Martin Jr. and Andrei Serjantov, editors, *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, pages 35–50. Springer, 2004.

[180] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications, Second Edition*. Wiley, 2006.

[181] Paul Syverson. *"Sleeping dogs lie in a bed of onions but wake when mixed"*. 4th Hot Topics in Privacy Enhancing Technologies (HotPETs 2011), 2011.

[182] George Danezis and Len Sassaman. *"Heartbeat traffic to counter (n-1) attacks: red-green-black mixes"*. In Sushil Jajodia, Pierangela Samarati, and Paul F. Syverson, editors, *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society, WPES 2003, Washington, DC, USA, October 30, 2003*, pages 89–93. ACM, 2003.

[183] Panoramix Project. https://panoramix.me/.

[184] Yawning Angel and George Danezis and Claudia Diaz and Ania Piotrowska and David Stainton. Katzenpost Mix Network End-to-end Protocol Specifica. https://github.com/katzenpost/docs/blob/master/specs/end_to_end.rst, .

[185] Yawning Angel and George Danezis and Claudia Diaz and Ania Piotrowska

and David Stainton. Katzenpost Mix Network Specification. https://github.com/katzenpost/docs/blob/master/specs/mixnet.rst, .

[186] Yawning Angel and Claudia Diaz and Ania Piotrowska and David Stainton. Katzenpost Mix Network Public Key Infrastructure Specification. https://github.com/katzenpost/docs/blob/master/specs/pki.rst.

[187] Yawning Angel and George Danezis and Claudia Diaz and Ania Piotrowska and David Stainton. Sphinx Mix Network Cryptographic Packet Format Specification. https://github.com/katzenpost/docs/blob/master/specs/sphinx.rst, .

[188] Nym Technologies. https://nymtech.net/.

[189] Nikita Borisov, George Danezis, and Ian Goldberg. *"DP5: A Private Presence Service"*. PoPETs, 2015(2):4–24, 2015. URL https://doi.org/10.1515/popets-2015-0008.

[190] Rahul Parhi, Michael Schliep, and Nicholas Hopper. *"MP3: A More Efficient Private Presence Protocol"*. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*, pages 38–57. Springer, 2018.

[191] Roger Dingledine and Nick Mathewson. *"Anonymity Loves Company: Usability and the Network Effect"*. In *5th Annual Workshop on the Economics of Information Security, WEIS 2006, Robinson College, University of Cambridge, England, UK, June 26-28, 2006*, 2006.

[192] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. *"Onion Routing"*. Commun. ACM, 42(2):39–41, 1999. URL https://doi.org/10.1145/293411.293443.

[193] Dakshi Agrawal and Dogan Kesdogan. *"Measuring Anonymity: The Disclosure Attack"*. pages 27–34, 2003.

[194] Laisen Nie, Dingde Jiang, and Zhihan Lv. *"Modeling network traffic for traffic matrix estimation and anomaly detection based on Bayesian network in cloud computing networks"*. Annales des Télécommunications, 72(5-6): 297–305, 2017. URL https://doi.org/10.1007/s12243-016-0546-3.

[195] D.P. Heyman and M.J. Sobel. *"Superposition of renewal processes"*. Stochastic Models in Operations Research: Stochastic Processes and Operating Characteristics, 2004.

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

[197] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. *"XFT: Practical Fault Tolerance beyond Crashes"*. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operat-*

ing Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016, pages 485–500, 2016.

[198] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[199] Don Johnson, Alfred Menezes, and Scott A. Vanstone. *"The Elliptic Curve Digital Signature Algorithm (ECDSA)"*. Int. J. Inf. Sec., 1(1):36–63, 2001. URL https://doi.org/10.1007/s102070100002.

[200] Ralph C. Merkle. *"A Digital Signature Based on a Conventional Encryption Function"*. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, pages 369–378. Springer, 1987.

[201] Danny Dolev and H. Raymond Strong. *"Authenticated Algorithms for Byzantine Agreement"*. SIAM J. Comput., 12(4):656–666, 1983. URL https://doi.org/10.1137/0212045.

[202] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[203] Victor Shoup. *"Practical Threshold Signatures"*. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, pages 207–220. Springer, 2000.

[204] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. *"Robust Threshold DSS Signatures"*. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, pages 354–371. Springer, 1996.

[205] Mihir Bellare, Juan A. Garay, and Tal Rabin. *"Distributed Pseudo-Random Bit Generators - A New Way to Speed-Up Shared Coin Tossing"*. In James E. Burns and Yoram Moses, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, pages 191–200. ACM, 1996.

[206] George Danezis and Prateek Mittal. *"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*. The Internet Society, 2009.

[207] Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. *"A Reputation System to Increase MIX-Net Reliability"*. In Ira S. Moskowitz, editor, *Information Hiding, 4th International Workshop, IHW*

2001, Pittsburgh, PA, USA, April 25-27, 2001, Proceedings, pages 126–141. Springer, 2001.

[208] Roger Dingledine and Paul F. Syverson. *"Reliable MIX Cascade Networks through Reputation"*. In Matt Blaze, editor, *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*, 2002.

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

[210] Oded Goldreich and Rafail Ostrovsky. *"Software Protection and Simulation on Oblivious RAMs"*. J. ACM, 43(3):431–473, 1996. URL https://doi.org/10.1145/233551.233553.

[211] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. *"Path ORAM: an extremely simple oblivious RAM protocol"*. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310. ACM, 2013.

[212] Yanbin Lu and Gene Tsudik. *"Towards Plugging Privacy Leaks in the Domain Name System"*. In *IEEE Tenth International Conference on Peer-to-Peer Computing, P2P 2010, Delft, The Netherlands, 25-27 August 2010*, pages 1–10. IEEE, 2010.

[213] Haya Shulman. *"Pretty Bad Privacy: Pitfalls of DNS Encryption"*. In Gail-Joon Ahn and Anupam Datta, editors, *Proceedings of the 13th Workshop on Privacy in the Electronic Society, WPES 2014, Scottsdale, AZ, USA, November 3, 2014*, pages 191–200. ACM, 2014.

[214] Hannes Federrath, Karl-Peter Fuchs, Dominik Herrmann, and Christopher Piosecny. *"Privacy-Preserving DNS: Analysis of Broadcast, Range Queries and Mix-Based Protection Methods"*. In Vijay Atluri and Claudia Díaz, editors, *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pages 665–683. Springer, 2011.

[215] Ian Goldberg. *"Improving the Robustness of Private Information Retrieval"*. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 131–148. IEEE Computer Society, 2007.

[216] Casey Devet, Ian Goldberg, and Nadia Heninger. *"Optimally Robust Private Information Retrieval"*. In Tadayoshi Kohno, editor, *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 269–283, 2012.

[217] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[218] B. Laurie. *"Apres-a system for anonymous presence"*. Technical report, 2004. http://apache-ssl.securehost.com/apres.pdf.

[219] A. Broder and M. Mitzenmacher. *"Network applications of Bloom filters: A survey"*. Internet mathematics, 2004.

[220] Wassily Hoeffding. *"Probability inequalities for sums of bounded random variables"*. Journal of the American statistical association, 1963.

[221] Raphael R. Toledo, George Danezis, and Ian Goldberg. *"Lower-Cost epsilon-Private Information Retrieval"*. CoRR, abs/1604.00223, 2016. URL http://arxiv.org/abs/1604.00223.

[222] Yo App. https://en.wikipedia.org/wiki/Yo_(app).

[223] "Yo App warns Israeli citizens of missile strikes". https://www.bbc.co.uk/news/technology-28247504.

[224] Google. *"Google Transparency Report - Making the web safer"*. Technical report, 2014. https://bit.ly/1A72tdQ.

[225] Thomas Gerbet, Amrit Kumar, and Cédric Lauradoux. *"A Privacy Analysis of Google and Yandex Safe Browsing"*. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016, Toulouse, France, June 28 - July 1, 2016*, pages 347–358. IEEE Computer Society, 2016.

[226] Sebastiano Di Paola and Dario Lombardo. *"Protecting against DNS Reflection Attacks with Bloom Filters"*. In Thorsten Holz and Herbert Bos, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 8th International Conference; DIMVA 2011, Amsterdam, The Netherlands, July 7-8, 2011. Proceedings*, pages 1–16. Springer, 2011.

[227] Shahabeddin Geravand and Mahmood Ahmadi. *"Bloom filter applications in network security: A state-of-the-art survey"*. Computer Networks, 57(18): 4047–4064, 2013. URL https://doi.org/10.1016/j.comnet.2013.09.003.

[228] Tor Metrics. https://metrics.torproject.org/.

[229] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. *"Summary cache: a scalable wide-area web cache sharing protocol"*. IEEE/ACM Transactions on Networking, 8(3):281–293, 2000. URL https://doi.org/10.1109/90.851975.

[230] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. *"Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers"*. In *26th Annual Network*

and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019. The Internet Society, 2019.