PRIVACY-PRESERVING SYSTEMS FOR MEASURING ANONYMITY NETWORKS

A Dissertation
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements of the
degree of Doctor of Philosophy
in Computer Science

By

Akshaya Mani, M.Sc.

Washington, DC
August 27, 2019

PRIVACY-PRESERVING SYSTEMS FOR MEASURING ANONYMITY NETWORKS

Akshaya Mani, M.Sc.

Thesis Advisor: Micah Sherr, Ph.D.

Abstract

Anonymity has become an important privacy requirement in today's internet. This is clearly evident from the upsurge in the use of overlay anonymity networks in the past few decades. Though anonymity networks have gained popularity and are widely used for enhancing online privacy, their usage is not well understood. Surprisingly, we know very little about who their users are, what they are being used for, and how they are being used/abused. This is because conducting traditional usage measurements on the live anonymity network can be ineffective or, worse, may risk the anonymity (and potentially safety) of the network's users. In fact, many existing attempts to measure anonymity networks (*e.g.*, Tor) have been highly criticized for unsafe research practices.

This thesis first explores the much simpler and privacy risk-free problem of measuring the *behavior* of anonymity networks. This does not involve any privacy risk as it entails comparing the results of traffic fetched via direct communication (*e.g.*, with a webserver) with traffic fetched through the anonymity network. We conduct a comprehensive study of the Internet's open proxies, that are often used as an alternative for achieving a weak form of sender anonymity. Our experiments show that misbehavior abounds on the Internet's open proxies and they achieve only a limited degree of anonymity. As a point of comparison, we also compare the level of manipulation observed when the content is fetched over an anonymity network, such as,

Tor. We find no instances in which Tor exit relays misbehave, suggesting that Tor offers a far more reliable and trustworthy form of anonymous communication.

We next address the much more challenging problem of understanding anonymity networks' usage. This involves gathering usage statistics (*e.g.*, client information, sites visited, *etc.*), which are sensitive and endanger the privacy (and safety) of the network's users. In this thesis, we develop efficient, safe and privacy-preserving measurement systems for anonymity networks (primarily Tor) that are both practical and scalable. Moreover, recording such usage statistics in volunteer-operated anonymity networks such as Tor, introduces additional privacy risks wherein relay operators can be compelled to release logs through a legal order or an extra-legal compulsion. Our measurement systems not only provide resistance against well-studied compromise and correlation attacks, but also against such "compulsion" attacks.

Finally, we deploy one of our measurement systems on the live Tor network and show that many of the Tor usage measurements of greatest interest (*e.g.*, the number of unique client IPs or visible hidden services, client distribution and diversity, popularity of ASes, *etc.*) can be done safely and practically. Moreover, our findings can aid in developing better Tor traffic models that would improve Tor's overall performance and security. The Tor Project's existing measurement methods (using heuristics) can also benefit in several ways from the security and privacy properties of our systems.

Acknowledgments

Many people have been instrumental in my success and I would like to convey my heartfelt thanks to them. My family, friends, teachers and mentors have moulded me into the person I am over the years. I sincerely regret being unable to name each and every one of them in this acknowledgement.

First and foremost, none of this would have been possible without my father, Mani Krishnan and my mother, Latha Mani. Their perennial encouragement and support helped me dream bigger and attain great heights. My dear sister, Aishwarya Mani and brother-in-law, Vaibhav Goel have also been a strong source of support during tough times, and I thank them deeply for it.

Even before I started at Georgetown, the mentorship of former professors and advisors such as Dr. Pandu Rangan, Dr. Anitha Renganathan, and Dr. R. Natarajan sowed the seeds of academic vigor and inquiry in my mind. I'm much obliged to them for making me realize my passion for research and the need to pursue a doctoral degree.

I would not be writing this thesis if not for the steadfast guidance of my advisor, Dr. Micah Sherr. His wise words and upbeat mentality have guaranteed that I stayed on course and successfully completed my research. As someone who prefers to go the extra mile, he always placed my career and personal goals at the forefront and motivated me to pursue them. Micah has been a role model - both as an academic and as a person. I am deeply indebted to him and will genuinely miss being under his tutelage.

I've also been privileged to work with Dr. Aaron Johnson and Dr. Sadia Afroz, stellar researchers who set great examples for me through their work. My experience with them has been nothing short of inspirational.

Along with the three people mentioned above, Dr. Clay Shields and Dr. Kobbi Nissim have also graciously agreed to be a part of my thesis committee. My sincere thanks to all of them for providing me valuable feedback and being a part of this crucial process, which is the culmination of my five years of diligence.

I am also very delighted to acknowledge and thank my co-authors and collaborators - Dr. Ellis Fenske, Dr. Rob Jansen, Dr. Tavish Vaidya, and Dr. Tim Wilson-Brown. Being around some of the sharpest minds in the field has made me hold myself up to high standards and accelerate my learning. My peers in the department Tavish Vaidya and Mohammad Zaheri made the journey all the more enjoyable, with our shared adventure of the roller-coaster ride that the PhD program is.

And, where would I be without family and friends? I consider myself incredibly blessed to have received a lot of love and support from them, especially during stressful times. As an international student, I am very fortunate to have had friends who were my family away from home. I felt nothing short of constant exhilaration around them.

Last but not the least, I am very grateful to my husband Aravind Gopalakrishnan, who stood patiently by me - day or night, rain or shine - in the final months of my PhD. I want to thank him for always being there for me.

The research conducted in the process of developing this dissertation was partially funded from National Science Foundation (NSF) grants CNS-1064986, CNS-1149832, CNS-1204347, CNS-1527401, and CNS-1718498. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the NSF.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid growth of the Internet, anonymity became an important privacy requirement [28, 71, 86, 117, 120]. However, the Internet was not designed for anonymous communication. Much like a physical envelope, IP packets contain information about the communicating parties and the routing protocols leave a trail as the data is transmitted across the internet.

A single-hop proxy can be used to mask the sender's IP address before relaying the users' traffic to their destinations. Although this achieves a weak form of anonymity, it comes at a heavy price – the users must trust the proxy server unconditionally. The proxy server knows both the end-points of the communication and is well positioned to eavesdrop, perform man-in-the-middle (MitM) attacks, inject spurious ads, or even record any traffic passing through it. This also makes the proxy extremely vulnerable to legal compulsion attacks, wherein the proxy operator can be subpoenaed to reveal the sender's identity.

Chaum [40] introduced the concept of overlay anonymity networks that operate at the application-layer. These networks hide a sender's identity by decoupling network location from routing using cryptography. They are based on the idea that data sent from the sender to the destination is wrapped up in layers of

encryption and relayed through one or more proxy servers (or "Mixes"). A Mix decrypts, reorders, and delays data from several end-to-end connections, thereby concealing not only the sender's IP address but also the path between the sender and the destination. Since then, a wide variety of such anonymity systems have been proposed [41, 46, 49, 60, 72, 92, 94, 118, 121, 134, 137, 140].

Although anonymity networks have gained popularity since then, some users prefer open proxies (*i.e.*, single-hop proxy servers that are accessible by any Internet user) purportedly for their high speed and ease of use. A quick search for "open proxies" on any search engine yields several websites devoted to maintaining large lists of available open proxies. Moreover, in contrast to some (but not all) anonymity systems, they require minimal configuration changes (*e.g.*, adjusting "network settings") rather than the installation of special software. However, some open proxies may not provide any anonymity at all and expose the sender to their destinations by inserting an additional header that contains the sender's IP address. Furthermore, previous research [115, 133] show that misbehavior abounds on the Internet's open proxies and that the use of such proxies carries substantial risk. In comparison to open proxies, anonymity services often relay traffic through multiple proxy servers (or "relays") and offer far more reliable and safe form of anonymous communication.

Tor [53] is the most popular anonymity network with an estimated 2.7 million daily users [130], as of August 2019. Other well-known anonymity networks include JonDonym [4] (formerly JAP [90]) and i2p [3]. Though anonymity networks are reliable and widely used for enhancing online privacy, surprisingly their usage is not well understood. We know very little about who is using them and for what purpose (*i.e.*, legitimate or illicit). Gathering usage statistics such as information about clients, traffic patterns, sites visited, *etc.*, can help us to better understand how these networks are being used and/or abused. However, naïvely recording these

sensitive statistics at the relays can endanger the privacy (and safety) of the network's users. Even if the entities collecting this information are honest, the data can still be exposed (*e.g.*, through machine compromise or compulsion from government authorities), posing serious privacy risks. Furthermore, published statistics even in highly aggregated form could be combined with background knowledge (*e.g.*, when an e-mail was sent, ISP logs showing user accesses to the anonymity network, *etc.*) to deanonymize users [47].

1.1 Challenges in Measuring Anonymity Networks' Usage

There are three main characteristics that make anonymity networks particularly unfriendly platforms for conducting (ethical) empirical studies:

1. *Privacy*: Experiments that capture users' communications can compromise the anonymity of the users who depend on the network to hide their identities.

2. *Integrity*: These are often volunteer-operated networks where anyone can spawn a malicious node. Therefore, measurement studies should consider a threat model in which the adversary attempts to manipulate measurements to further their goals.

3. *Security*: A measurement system should not provide a new attack surface for disrupting or otherwise manipulating the anonymity network.

Ideally, only a privacy-preserving measurement system that gathers user statistics *privately* can be used – that is, nothing should be revealed other than the data of interest. But in this thesis, we relax this rigid requirement in favor of practicality. That is, we aim to develop privacy-preserving techniques that provide useful measurements while offering quantifiable privacy risks.

In the reminder of this thesis, we will focus on Tor, the most widely used and well-studied anonymity service.

1.2 Our Current Understanding of Tor

The Tor network has been in operation since 2004 [53], enabling millions of daily users [130] to anonymously communicate through the Internet. Despite its relative longevity, growing popularity and prominence as a privacy-enhancing communication service, the knowledge about its real-world usage is unfortunately incomplete, outdated, and sometimes even contradictory. According to the Tor Project, Tor is used by ordinary people for private browsing, journalists for safe reporting of news, law enforcement officers for browsing questionable websites, and whistleblowers for reporting wrong doings [131]. However, according to many others [109], Tor is used by variety of criminals to avoid surveillance.

The study by McCoy et al. [105] is perhaps the earliest attempt at understanding how Tor is used. But its findings were based on surreptitiously capturing and analyzing the users' real-time traffic flows through a Tor relay. This approach was criticized both on ethical and legal grounds [124, 125], due to the "network snooping" and privacy (and potentially security) risks involved in recording unobfuscated network traces in an anonymity network such as Tor.

Moreover, experiments that capture users' communications are antithetical to the Tor Project's mission. For this reason, the Tor Metrics Portal [130], a Tor operated data repository that archives information about the anonymity network's size, makeup, and capacity, provides only a limited number of statistics, many of which are based on heuristics of unknown accuracy. This technique to publish statistics

from Tor in a "safe way" was originally proposed by Loesing et al. [99] in 2010. However, they do not provide any formal security or privacy guarantees.

The debate [119] between supporters of the Tor network and the Cloudflare CDN further highlights the lack of a clear understanding of how Tor is used. For a brief period of time Cloudflare forced Tor users to complete cumbersome CAPTCHAs before accessing any of the vast number of Cloudflare's hosted sites. The CTO of Cloudflare cited the large portion of attack traffic originating from Tor exit relays as the principle motivation behind this setup. The Tor Project publicly refuted this claim [116] by questioning the accuracy of Cloudflare's measurements, and pointed to a report by Akamai [14] showing that Tor traffic had a similar proportion of attack traffic as the regular (non-anonymized) Internet.

On the bright side, Cloudflare recently launched the Cloudflare onion service that allows legitimate Tor users (using Tor browser 8.0 or higher) to access Cloudflare websites without encountering CAPTCHAs. While this is a significant win for Tor, it still does not address the bigger concern of how Tor is being used/abused.

Similar research by Biryukov et al. showed that an enormous percentage of connections to Tor Hidden Services are to websites serving child pornography [31]. This was again disputed by the maintainers of the Tor Project [103].

1.3 Safe Tor Measurements with Differential Privacy

Differential Privacy [57] is a privacy definition that provides provable and quantifiable privacy guarantees regarding the information that is leaked when publishing statistics from a database. Differentially private techniques have the potential to ensure that published Tor statistics look nearly the same, regardless of whether any single Tor user's information is included in (or excluded from) the published results.

For instance, if we collect the number of visits through Tor to *google.com*, differential privacy guarantees that for any adversary, the result when a target user visits *google.com* is indistinguishable from the result when the user does not. More importantly, differential privacy makes no assumption on the knowledge or computational capability of the attacker.

Many privacy-preserving data aggregation systems based on differential privacy mechanism have been proposed [34, 42, 43, 62, 64, 80, 108]. The most relevant of these are the PrivEx [62] and the PrivCount [80] systems that were particularly designed for gathering user statistics from anonymity networks such as Tor. PrivEx has two variants – one based on secret sharing and the other on distributed decryption – that use differential privacy to collect statistics from Tor in a privacy-preserving way. PrivEx has (justifiably) garnered significant attention from the privacy-enhancing technologies community since then, and has been promoted as an example of a tool that enables safe data collection on Tor [132]. PrivCount extends the secret sharing variant of PrivEx to make it suitable for a small-scale research deployment. It allows multi-phase iterative measurements and expands the privacy notion of PrivEx to simultaneously handle multiple and diverse Tor statistics with an optimal allocation of the ϵ privacy budget. Both PrivEx and PrivCount do not leak any private information on an individual Tor user, as long as there is at least one honest data collector (*i.e.*, Tor relay) and at least one honest-but-curious aggregator.

Both PrivEx and PrivCount provide resistance to the so-called "compulsion attacks" in which a relay operator can be compelled to release information to a (potentially totalitarian) government in a volunteer-operated network such as Tor. Gathering statistics obviously involves logging statistics at relays, which inherently increases privacy risks since these statistics would not be collected otherwise. For

example, Tor does not currently log client connections to guards; queries which ask for the number of client connections thus may impose additional privacy risks.

While both PrivEx and PrivCount measurement systems significantly raise the bar for safe Tor measurements, there are two main limitations:

1. *Integrity*[1]: They are not immune to manipulation. In particular, a single malicious relay operator can drive the aggregate of the statistic being computed towards an arbitrary value of its choosing, without risking detection. To demonstrate this, we made a trivial modification to the PrivEx source code (as provided by Elahi et al. [62]). Consistent with their work, we consider a query that aggregates the number of visits destined to 15 particular websites. This query can be used in gaining insights such as the list of most popular websites among Tor users. In our experiment, we use the same default parameters as is included in the PrivEx source code. However, we modify the behavior of a single relay to falsely claim that it has observed 1000 visits to website #2.

   Figure 1.1 shows the result of this manipulation. The actual distribution of website visits is shown in hollow green and indicates that website #6 is, by far, the most popular site visited via Tor. The modified (and noised) distribution as reported by PrivEx, shown in solid red, paints a different picture: according to the results of the query, website #2 appears to similarly be popular amongst Tor users.

2. *Count-distinct Problem*: Both PrivEx and PrivCount lack the ability to perform counts of distinct private values across data collectors (*i.e.*, Tor relays).

[1] In this context, by *integrity* we mean the validity or trustworthiness of the measurement results being computed. An adversary (operating a malicious relay) must not be able to manipulate the results to further their goals (see §1.1).

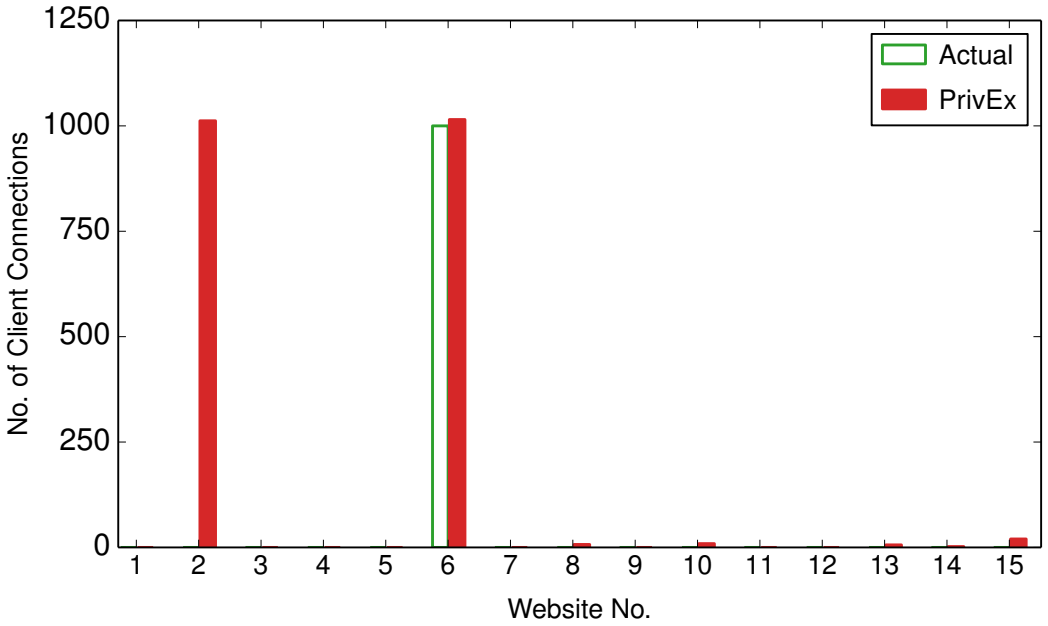

Figure 1.1: Manipulating PrivEx. The number of client connections (y-axis) for 15 different websites (x-axis) as actually occurred ("Actual") and as reported by PrivEx with noise added ("PrivEx"). A single malicious relay falsely reports that it observed 1000 connections to website #2.

> More formally, this problem is referred to as the *private set-union cardinality* estimation problem. For example, both PrivEx and PrivCount can answer the question – *how many client connections were observed on Tor?* – but cannot discern what fraction of the result consists of unique clients. That is, they cannot answer the (perhaps more useful) question of *how many unique clients were observed on Tor?*

The lack of integrity guarantees and the ability to estimate unique count across a set of distributed private datasets is problematic for many types of queries that would be useful for Tor. For one, Tor is a volunteer-operated network and operating a malicious Tor relay is not difficult. Indeed, there have been several instances [139] in which relay operators have been found to behave maliciously. Moreover, many

useful client, exit, and onion statistics such as the number of unique client IPs, websites visited, onion addresses, *etc.* are count-distinct problems. Therefore a measuring system for anonymity networks like Tor must consider integrity protections and private set-union cardinality estimation, in addition to privacy guarantees for useful statistics gathering.

1.4 Summary

Before analyzing techniques for measurement of the usage of anonymity networks, this thesis first explores the behavior of anonymity networks themselves. We present a broad study of the Internet's open proxies that examines the prevalence of previously undisclosed attacks, and relate its findings to a similar study (which we conduct) on Tor exit relays. Our results demonstrate that Tor offers a far more trustworthy and safe form of anonymous communication.

We then propose two new tools – HisTorϵ and Private Set-Union Cardinality (PSC) – based on (ϵ, δ)-differential privacy to gather statistics from Tor in an efficient, safe and privacy-preserving manner. Like existing schemes [62, 80], both HisTorϵ and PSC provide resistance against compulsion attacks. Additionally, HisTorϵ provides strong integrity guarantees, and PSC computes count of unique items across distributed private data owners (*e.g.*, Tor relays). Though we designed these systems primarily for Tor, they have wide applicability and can be in principle used to collect statistics privately from other anonymity networks such as JonDonym [4] and i2p [3].

For our final contribution, we enhance PSC to support the measurement of unique domains, client IPs, and onion sites. We then perform a large study of user behavior on the live Tor network and describe the tactics used for selecting appro-

priate privacy parameters. Moreover, we analyze a number of client measurement results obtained using an enhanced PrivCount deployment to better understand Tor user behavior (such as who uses Tor and from where they access the network). Furthermore, we've described new methods for extrapolating our local observations to whole-network wide statistics. Through these techniques ensure individual Tor users' safety, while maintaining a high degree of accuracy.

1.5 Research Questions

In more detail, this thesis explores the following research questions:

- **How can we design private statistics gathering mechanisms that are resilient to adversarial manipulation?** Previous research [62, 80] does not provide integrity guarantees – even a single malicious relay can drive results to an arbitrary value (see Figure 1.1). We achieve robustness by forgoing general counting queries (as supported by previous work) in favor of supporting more robust binning queries. Conceptually, all relays (malicious or not) can contribute at most "1" to each bin, thus strictly bounding the influence over the total aggregate. The proposed statistics gathering system is generic (*i.e.*, not limited to Tor) and can be used for any anonymity service.

- **How can we compute the distinct-count across all relays?** Existing privacy-preserving data aggregation systems for Tor [62, 80] only support counting queries. However, as previously discussed, they lack the ability to collect many other useful Tor statistics such as the number of unique Tor users, the number of unique destinations visited, *etc.* We propose PSC, a protocol for aggregating the count of unique items across a set of distributed private datasets. The usage of this protocol is not limited to Tor, and can be used to gather statistics

in a privacy-preserving way from any distributed system such as anonymity networks, Chord [127], or the web.

- **How to protect the privacy of an individual Tor user while conducting live Tor network measurements with PSC?** We adapt the action bound approach developed by Jansen and Johnson [80]. This method bounds the maximum amount of network activity that can be protected using differential privacy, for a "reasonable" Tor user over a measurement epoch (*i.e.*, duration of the measurement). This is required as protecting hypothetical users, whose individual activity contributes major portion of the network activity, would yield inaccurate results due to a high fraction of differentially private noise.

  Based on how certain types of reasonable Tor user actions translate into observable activity (measured using PSC) on the live Tor network during an epoch, we derive new action bounds. Fortunately, these bounds constitute only a small fraction of the total network activity, thereby providing accurate network measurements while protecting the privacy of individual Tor users.

- **How to infer Tor network wide statistics from our measurement results?** Extrapolating unique counts from our sample to the entire network can be a challenge. Unlike with standard counts, we need to know if the same items in our sample are seen elsewhere or not. First we come up with models based on how Tor works and then use simulation to extrapolate local confidence intervals (for the observations at our relays) for the network-wide count.

1.6 Contributions

The main contributions of this thesis are:

- **Evaluation of the Internet's Open Proxies.** *(Chapter 3)* We evaluating the behavior of anonymity networks. We present an empirical study of the Internet's open proxies (that provide a weak form of sender anonymity), encompassing more than 107,000 publicly listed open proxies. We design a series of experiments to understand a broad range of proxy behaviors. We compare the results of traffic received via direct communication with traffic that traverses through proxies. Finally, we contrast the level of manipulation observed on open proxies against manipulation on anonymity networks such as Tor.

- **Private and Robust Statistics Collection for Tor.** *(Chapter 4)* We present HisTorϵ, a privacy-preserving statistics collection scheme based on (ϵ, δ)-differential privacy that is robust against adversarial manipulation. We formalize the security guarantees of HisTorϵ and demonstrate using historical data from the Tor Project that HisTorϵ provides useful data collection and reporting with low bandwidth and processing overheads.

- **Distributed Private Set-Union Cardinality.** *(Chapter 5)* We introduce PSC, a cryptographic protocol for efficiently aggregating the count of unique items across a set of data collectors privately – that is, without exposing any information other than the count. We present a proof-of-concept implementation and use it to demonstrate that PSC operates with low computational overhead and moderate bandwidth, and is thus suitable for distributed private measurements in real-time.

- **Understanding Tor Usage with PSC** *(Chapter 6)* We describe significant enhancements for PSC to support safe measurement on the live Tor network. We present a detailed measurement study of Tor using PSC, covering three major aspects of Tor usage: how many unique client IPs connect to Tor (including

their origin country), how many unique destinations do users most frequently visit, and how many unique onion services exist. We also propose methods to extrapolate the results of our measurement study for the entire Tor network.

1.7 Organization

The remainder of the thesis is presented as follows. In Chapter 2, we describe prior research in measuring anonymity networks. Chapter 3 presents a background on Tor, and differential privacy. In Chapter 4, we evaluate the behavior of the Internet's open proxies. A privacy-preserving statistics collection scheme, HisTorϵ, that is robust against adversarial manipulation is described in Chapter 5. Chapter 6 introduces PSC, a cryptographic protocol for efficiently computing private set-union cardinality in distributed systems. Chapter 7 presents a detailed measurement study of Tor using PSC. We conclude in Chapter 8.

Chapter 2

Related Work

In this chapter we present prior research that relates to the problem of measuring anonymity networks' usage (primarily Tor). This problem has not been explored much by the research community due to the privacy risks involved. We also review previous research that measures the *behavior* of anonymity networks by itself, which is a much simpler problem as there are no privacy risks of deanonymizing the network's users.

2.1 Measuring Behavior of Anonymity Networks

We first review prior research that focuses on measuring the behavior of anonymity networks.

Open Proxies. Open proxies are often used as an alternative to anonymity systems purportedly for their high speed and ease of use. We are of course not the first to investigate the behavior of open proxies on the Internet. Scott et al. [123] provide insights about the use, distribution, and traffic characteristics of open proxies on the Internet. However, they do not investigate malicious behavior by open proxies.

Other research has investigated the misuse of proxy servers. Of note, Steding-Jessen et al. [126] deploy low-interaction honeypots to explore the (ab)use of open

proxies to send spam. Tyson et al. [135] study HTTP header manipulation on the Internet by various open proxies and discuss various observed factors affecting HTTP header manipulation. Tsirantonakis et al. [133] look at content manipulation by examining roughly 66K open HTTP proxies on the Internet. Their work provides an in-depth analysis of different types of malicious behavior exhibited by proxies by injecting Javascript code.Their analysis shows that proxies inject Javascript aimed at tracking users, stealing user information, fingerprinting browsers, and replacing ads.

In this thesis, we provide an in-breadth analysis of 107K unique open proxies by evaluating their availability, performance, and behavior across a large spectrum of potential attacks (see Chapter 4). We look at the manipulation of content by open proxies, not only through the injection of Javascript, but also by modification of HTML content and through TLS MitM. Additionally, unlike existing work, we evaluate the behavior of open proxies based on different client locations.

Tor. Tor [53] provides anonymous TCP communication by routing user traffic through multiple relays (typically three) using layered encryption (refer §3.1). The first relay in the path (or circuit) is the guard relay, and the final relay through which traffic exits is the exit relay. The original data transmitted is visible only at the exit relays. Therefore, unless e2e encryption is used, data can be eavesdropped by malicious or compromised exits. There is a large body of related prior work that measures malicious behavior by Tor exits [38, 105, 139]. These studies found evidence of malicious behavior such as interception of credentials, *etc.*, by a small fraction of Tor exit relays, especially when the bait traffic was not encrypted end-to-end.

As a part of this thesis, we perform a similar measurement study of the behavior of Tor exit relays. Surprisingly, we do not find any evidence of content manipulation or TLS MitM by Tor exits.

2.2 Measuring Tor Usage

We now review the much less explored problem of measuring the usage of the Tor anonymity network.

Preliminary Attempts. McCoy et al. [105] performed one of the first studies that attempted to discern *how* users used Tor. There, the authors performed packet capture at a small subset of Tor ingress and egress points to determine the distributions of user locations and exit traffic by port (*i.e.*, by application). Chaabane et al. [37] used similar methods to examine BitTorrent behavior on Tor. Approaches such as these that collect unobfuscated and potentially sensitive network traces have been criticized on both ethical and legal grounds [124, 125]. In particular, Soghoian [125] has called for improved community standards for research on Tor.

Tor Metrics Portal. Loesing et al. [99] reiterate the need for ethical measurement studies, and motivate the need for performing statistical analysis of the Tor network. The authors promote privacy-preserving statistics gathering as a means to identify trends in the Tor network, to detect performance bottlenecks, to discover attacks against the network, and to better understand how Tor is being blocked and censored in various regions of the Internet [99]. They propose techniques to publish statistics from Tor in a "safe way" using heuristics of unknown accuracy. However, they do not provide any formal security or privacy guarantees.

The Tor Metrics Portal [130] is a data repository operated by the maintainers of Tor. It allows researchers to access aggregate and longitudinal data about the Tor network since 2010, with directory data spanning back to May 2004. To date, it uses the approach proposed by Loesing et al. to gather some usage statistics, including the number of Tor users.

We note that now Tor Project also promotes the use of differentially private statistics collection – by use of differentially private techniques to publish per-relay onion-service statistics [77].

Privacy-Preserving Measurements. There has been an increasing effort in developing techniques to *safely* measure the Tor network. Several measurement techniques have been proposed for Tor that use differential privacy to protect user privacy. Here, the goal is to produce useful aggregate statistics about the Tor network, while providing quantifiable privacy guarantees.

Elahi et al. [62] were the first to propose the use of differential privacy to privately collect sensitive traffic statistics from Tor. Specifically, they propose two schemes variants – one based on secret sharing and another that uses distributed decryption – known collectively as PrivEx. PrivEx provides strong, measurable privacy protection even against compromised participants.

PrivCount [80] extends the PrivEx secret-sharing variant by supporting repeatable measurement phases. PrivCount allows multi-phase iterative measurements and expands the privacy notion of PrivEx to simultaneously handle multiple and diverse Tor statistics with an optimal allocation of the ϵ privacy budget.

However, both PrivEx and PrivCount do not provide integrity guarantees, and as we demonstrate in §1.3, even a single malicious relay can cause inaccurate results. While such an attack may be outside of the threat models envisioned by PrivEx and PrivCount, the attack points to the need for statistics gathering mechanisms that are both privacy-preserving and resilient to manipulation. In this thesis, we present HisTorϵ, a differentially private statistics gathering system, that additionally provides strong integrity guarantees about the influence of malicious DCs.

Moreover, though both PrivEx and PrivCount can aggregate counts across relays, they cannot compute *unique* counts, which is the functionality provided by

PSC, an (ϵ, δ)-differentially private statistics gathering system presented in Chapter 5.

Corrigan-Gibbs and Boneh proposed Prio [45], a privacy-preserving system for computing any aggregate function on a set of private inputs. Like PrivEx and PrivCount, Prio also provides strong privacy guarantees. And like HisTorϵ (described in this thesis), it maintains robustness in the presence of an unbounded number of malicious clients,

Understanding Onion Services. Tor allows services to hide their locations through the use of *onion services* [129]. As a special (but common) case, when the onion service corresponds to a hidden website, we refer to it as an onionsite.

Biryukov et al. demonstrate how the design of onion services allows an attacker to gauge the popularity of an onion service and potentially deanonymize it [30]. However, their envisioned attacks are not suitable for conducting privacy-preserving measurements. Goulet et al. [77] describe the benefits and privacy risks of statistics collection for onion services. They conclude with suggestions for privacy-preserving statistics collection, including the use of differential privacy. Other existing work[31, 114] also perform empirical measurements of Tor's onion services. They perform content analysis also attempt to quantify the number of onion services published. However, they do not employ any privacy-preserving techniques. Finally, Jansen et al. show how to use PrivCount to safely measure the Tor network to discover the popularity of onion services [81]. Their work focused on traffic analysis attacks and the popularity study of a single social networking onion service.

In this thesis, we perform a similar measurement study on onion sites – by operating Tor relays and observing HSDir lookups – but also protect user privacy by using (ϵ, δ)-differentially private techniques.

Chapter 3

Background

In this chapter, we review some preliminaries that are necessary to understand the remainder of the thesis.

3.1 Tor

Tor is a network of volunteer-operated routers that enhances privacy by separating network identity from network location [53]. The Tor client proxies a user's TCP connections through a series of randomly selected relays, which collectively form a circuit. Several TCP connections may be multiplexed over the same circuit. The first hop on this circuit – the ingress point to the Tor network – is usually a *guard*, a Tor relay that is relatively stable and is deemed to have sufficient bandwidth for the task. As a notable exception, traffic may also enter the Tor network through bridge nodes, which are similar to guards but are not publicly advertised. Traffic exits the Tor network through exit relays, which establish TCP connections to the intended destination. Along the Tor circuit, cryptography helps conceal the actual sender and receiver – relays know only the previous and next hop along the anonymous path.

Tor is designed to be robust against a non-global adversary, meaning that it provides protections against an adversary who cannot view all traffic on the network.

It is known to be vulnerable against traffic correlation attacks [86] in which an adversary can observe an anonymous connection as it enters and leaves the anonymity network. Here, using timing and flow analysis, the adversary can correlate the ingress and egress traffic and determine that they belong to the same traffic stream, thus deanonymizing both endpoints of the communication.

3.2 Differential Privacy

Differential privacy [57] is a privacy definition that offers provable and quantifiable privacy of database queries. Differential privacy guarantees that the query responses look nearly the same regardless of whether or not the input of any one user is included in the database. Thus anything that is learned from the queries is learned independently of the inclusion of any single user's data [85]. Several mechanisms have been designed to provide differential privacy while maximizing accuracy [32, 58, 107, 112].

Differential privacy is typically achieved by adding noise to query results. This noise is added in a controlled manner such that the result of the query is nearly the same for adjacent databases. More formally, an (ϵ, δ)-differentially-private mechanism is an algorithm $\mathcal{M}$ such that, for all datasets D_1 and D_2 that differ only on the input of one user, and all $S \subseteq \text{Range}(\mathcal{M})$, the following holds:

$$Pr[\mathcal{M}(D_1) \in S] \leq e^{\epsilon} \times Pr[\mathcal{M}(D_2) \in S] + \delta. \tag{3.1}$$

Crucially, differential privacy's formal guarantees are about the properties of the database mechanism rather than the computational capabilities or auxiliary knowledge of the adversary. In practice, differential privacy is achieved by adding noise to the result of some aggregate query. The parameter ϵ controls the tradeoff between

privacy and utility: a larger ϵ provides weaker privacy but more accurate results. The inclusion of δ relaxes the more rigid notion of ϵ-differential privacy and allows for more practical implementations.

However, as Jansen and Johnson [80] show, this type of user-level differential privacy is inherently difficult to achieve in case of anonymity network (such as Tor) measurements. This is because an adversary might be able to link together multiple observations to a specific user or worse bound the total amount of the user activity. Therefore, we provide event-level privacy Jansen and Johnson, *i.e.*, instead, we protect all users whose actions translate into a limited amount of network actions within a given length of time. Here, limiting the amount of network activity protected by differential privacy is crucial to produce accurate results, as otherwise we will end up protecting a hypothetical users (such as a bot) whose activity constitutes the majority of the network activity.

Differential privacy (user-level or event-level) is typically achieved by adding noise to query results. Dwork [57] prove that binomial noise, that is, the sum of n uniformly random binary values, provides (ϵ, δ)-differential privacy for queries that each user can affect by at most one when

$$n \geq \left(\frac{64 \ln(2/\delta)}{\epsilon^2} \right) \tag{3.2}$$

Eq. 3.2 yields the amount of binomial noise that is required to be added to the output of a query that each user can change by at most one. In this thesis, we use this binomial noise technique to achieve differential privacy.

Suppose there are c rows in the database, then the c rows and n noise vectors are shuffled and summed together. Letting b_i be the ith element (where $1 \leq i \leq c + n$), the (noised) aggregate result $\mathbf{z}$ is obtained as

$$\mathbf{z} = \sum_{i=1}^{c+n} b_i - \frac{n}{2} \tag{3.3}$$

Here, the $n/2$ term corrects for the expected amount of added noise.

In the following chapter, we first focus on measuring the behavior of anonymity networks (Chapter 4). We perform an extensive evaluation of the Internet's open proxies that are often used to achieve a weak form of sender anonymity. We compare the level of manipulation observed through open proxies to that when the content is fetched over Tor, the most widely used anonymity network.

Chapter 4

Evaluation of the Internet's Open Proxies

We begin by focusing not on the usage of an anonymity system, but rather on how the system itself behaves. Measuring the behavior of an anonymity system is a much simpler problem as it involves comparing the results of traffic received via direct communication (*e.g.*, with a webserver) with traffic that traverses through the anonymity network. Therefore, there is no privacy risk of deanonymizing the network's users.

This chapter presents a comprehensive study of the Internet's open proxies that are often used to achieve weak sender anonymity. The study comprises of more than 107,000 listed open proxies and 13M proxy requests over a 50 day period. Our results show that listed open proxies suffer poor availability – more than 92% of open proxies that appear on aggregator sites are unresponsive to proxy requests. While our results indicate that the majority of open proxies seemingly operate correctly (that is, they forward the traffic unimpeded), we find a surprisingly large number of instances of misbehavior. In particular, we discover numerous instances in which proxies manipulate HTML content, not only to insert ads, but also to mine cryptocurrency (*i.e.*, *cryptojacking*). Our study also discloses numerous attacks in which proxies conduct TLS man-in-the-middle (MitM) attacks. Finally, as a point of comparison, we conduct and discuss a similar measurement study of Tor, the most popular

```
GET http://neverssl.com/index.html HTTP/1.1
User-Agent: Chrome/62.0.3202.94

HTTP/1.1 200 OK
Date: Wed, 30 May 2018 14:28:02 GMT
Server: Apache
Content-Length: 12345
```

Figure 4.1: Example protocol interaction for HTTP proxy. Example HTTP requests (in blue) and HTTP responses (in red) for HTTP proxies.

anonymity service out there. Over the course of our nearly monthlong experiment in which we fetched files from every available Tor exit relay, we find no instance in which the requested content was manipulated. Our result suggests that Tor offers a more reliable and trustworthy form of anonymous communication.

4.1 Background: Open Proxies

Open proxy servers are unrestricted proxies that allow access for any Internet user. Open proxies are fairly prevalent on the Internet, and several websites are devoted to maintaining large lists of available open proxies. In contrast to VPNs and some (but not all) anonymity systems, open proxies are generally easy for users to use, requiring only minimal configuration changes (*e.g.*, adjusting "network settings") rather than the installation of new software.

There are various types of open proxies, with differing capabilities. Commonly, open proxies use HTTP as a transport mechanism. This has the benefit of supporting clients whose local firewall policies prohibit non-HTTP traffic (*e.g.*, as is sometimes the case on corporate networks).

We distinguish between two types of proxies that rely on HTTP. The first, which we refer to simply as *HTTP proxies*, allows clients to specify a fully-qualified URL – with any domain name – as part of a HTTP GET request to the proxy. The proxy parses the GET URL and if the requested URL is on a different host, the proxy makes its own request to fetch the URL and forwards the response back to the client. Conceptually, a HTTP proxy is a web server that serves pages that are hosted elsewhere. Example protocol interaction for HTTP proxy is shown in Figure 4.1.

A significant disadvantage of HTTP proxies is that it is incompatible with end-to-end (e2e) security protocols such as TLS, since it is the HTTP proxy (not the user's browser) that initiates the connection to the proxied URL. The inability to support fetching HTTPS URLs increasingly limits the use of HTTP proxies due to the growing adoption rate of HTTPS [65] on the web.

In contrast, *CONNECT proxies* use the HTTP CONNECT method [87] to establish e2e tunnels between the client (*i.e.*, the browser) and the destination webserver. CONNECT proxies allow the client to specify a host and port. The proxy then initiates a TCP connection to the specified target and then transparently forwards all future TCP data from the client to the destination, and vice versa. Importantly, CONNECT proxies forward traffic at the TCP level (layer 4) and are not limited to any particular application-layer protocol. CONNECT proxies support TLS/HTTPS connections since the browser can effectively communicate unencumbered to the destination webserver and perform an ordinary TLS handshake. Example protocol interaction for CONNECT proxy is provided in Figure 4.2.

Another type of open proxies, *SOCKS proxies*, use Socket Secure (SOCKS) protocol [96] (introduced in 1992 as a means to ease the configuration of network firewalls [89]) as a transport mechanism. The SOCKS proxy server listens for incoming TCP connections from a client. Once connected, the client can then tunnel

```
CONNECT www.acsac.org:443 HTTP/1.1
Host: www.acsac.org:443
Connection: keep-alive
User-Agent: Chrome/62.0.3202.94

HTTP/1.1 200 Connection Established
Connection: close
```

Figure 4.2: Example protocol interaction for CONNECT proxy. Example HTTP requests (in blue) and HTTP responses (in red) for CONNECT proxies.

TCP and/or UDP traffic through the SOCKS proxy to its chosen destination(s). Since SOCKS forwards arbitrary TCP and UDP traffic, clients can use end-to-end (e2e) security protocols (in particular, TLS/SSL) through SOCKS proxies. Currently, SOCKSv4 and SOCKSv5 servers are both deployed on the Internet, with the latter adding authentication features that enforce access control. In this chapter, we do not distinguish between SOCKSv4 and SOCKSv5, and use the general term *SOCKS proxies* to describe proxies running either version.

The network addresses (IPs and ports) of open proxies are listed on *proxy aggregator sites*. These sites also sometimes use the term *transparent*, meaning that the proxy supports at least e2e TCP tunneling; SOCKS and CONNECT proxies meet this criterion. As another dimension, aggregator sites also sometimes categorize proxies as *anonymous proxies*. These proxies purportedly do not reveal the client's IP address to the destination and provide weak form of sender anonymity.

4.2 Methodology and Experimental Setup

Our study of the Internet's open proxy servers has two main goals: (i) to measure the proxies' availability, and composition and (ii) to assess the degree to which proxies exhibit malicious behavior. We begin by describing the methodology used to perform our study.

We conducted our study over a 50 day period, beginning on 2018-04-12 and ending on 2018-05-31. Our measurement apparatus, described next, was installed on 16 locations (listed in Table 4.1); these included 15 geographically diverse AWS regions and an installation at Georgetown University. We term each instance a *client location*. Having multiple client locations allows us to determine whether proxies behave differently based on the network and/or geographic locations of the requesting client.

From each client location, an automated process performed the following steps once per day:

1. *Populate:* We collect and combine lists of advertised proxies from a number of proxy aggregator sites. We augment this list with the results of running ProxyBroker [44], an open source tool for finding open proxies. In all cases, proxies were listed by aggregator sites as tuples containing an IPv4 address and a TCP port number. The complete list of sources of proxies is listed in Table 4.2. We emphasize that the list of proxies is (re)fetched daily since, open proxies are subject to high levels of churn.

2. *Classify:* Next, from each client location, we attempt to connect to each proxy and, if successful, determine whether the proxy is a HTTP, CONNECT, or SOCKS proxy.

Table 4.1: Client locations in open proxies experimental setup.

Location	Downtime
California	—
Canada	—
Frankfurt	1 day
Ireland	—
London	—
Mumbai	—
Paris	—
Ohio	—
Oregon	—
São Paulo	—
Seoul	—
Singapore	—
Sydney	—
Tokyo	—
Virginia	—
Georgetown University	3 days
Count: 16	

Locations of clients and the number of days during the measurement period in which that client did *not* participate.

3. *Fetch:* Finally, we request several files (URLs) from the set of proxies that we were successfully able to classify.

In more detail, during the Fetch step, we retrieve an HTML page from each proxy that we were able to classify, using unencrypted HTTP connection. The URL for the HTML page is hosted on a web server at Georgetown University.

For CONNECT and SOCKS proxies (*i.e.*, the proxies that support TLS), we also request files over HTTPS from a properly configured (*i.e.*, with a valid certificate) web server running at Georgetown University. We additionally request HTML

Table 4.2: Sources of open proxies.

Proxy Aggregator	Proxies
clarketm [15]	6,343
multiproxy.org (all) [18]	1,524
multiproxy.org (anon) [19]	373
NordVPN [20]	29,194
ProxyBroker [44]	73,905
workingproxies.org [22]	1,250
xroxy [23]	345
Total (unique proxies)	107,034

A given proxy may be listed by more than one aggregator.

files from `https://revoked.badssl.com/` and `https://self-signed.badssl.com/` which, respectively, use revoked and self-signed certificates. The rationale for fetching content from sites with invalid certificates is discussed in §4.5. In all cases, we set the User-Agent HTTP request header to match that of Google Chrome version 62.0.3202.94 running on Mac OS X.

For each proxy request, we record whether the request completed. If we received a response from the proxy, we also record the HTTP status code and response string (*e.g.*, "`200 OK`") returned by the proxy, the size of the response, the content of the response, the MIME-type of the response (as determined by filemagic), the time-to-last-byte (TTLB) for receiving the response, the HTTP response headers, and (in the case of HTTPS requests) the certificate received.

Throughout this chapter, we use the term *expected content* to refer to the *correct* contents of the file and *unexpected content* to refer to content returned by a proxy that does not match the file indicated by the requested URL. A correctly functioning open proxy should thus return the expected content. As we explore in more detail

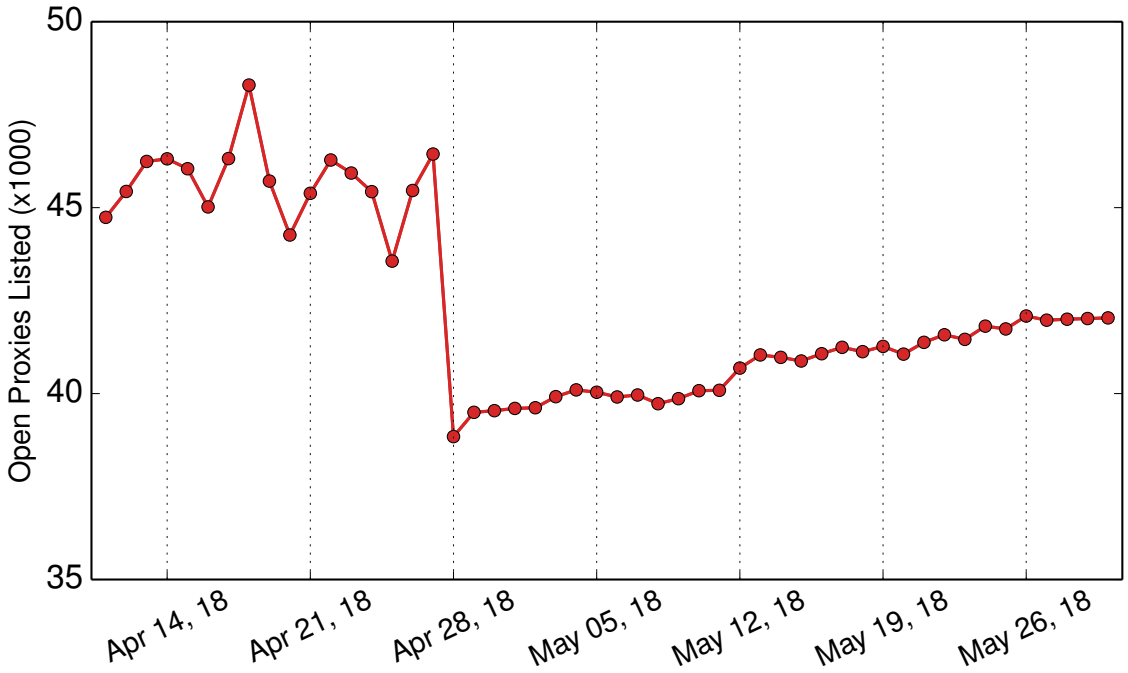

Figure 4.3: Open proxies publicly listed over time. Number of unique open proxies listed on aggregator sites, over time.

below, unexpected content does not necessarily indicate malicious behavior. For instance, unexpected content could be an HTML page indicating that the proxy is misconfigured or that the proxy requires user authentication.

We remark that a weakness of our study, and one that we share with studies that examine similar (mis)behavior in anonymity and proxy services [38, 133, 139], is that we cannot easily differentiate between manipulation that occurs at a proxy and manipulation that occurs somewhere along the path between the proxy and the destination. Our findings of misbehavior can be viewed as indications that a proxy should not be used, either because it was itself malicious or because its network location makes traffic routed through it routinely vulnerable.

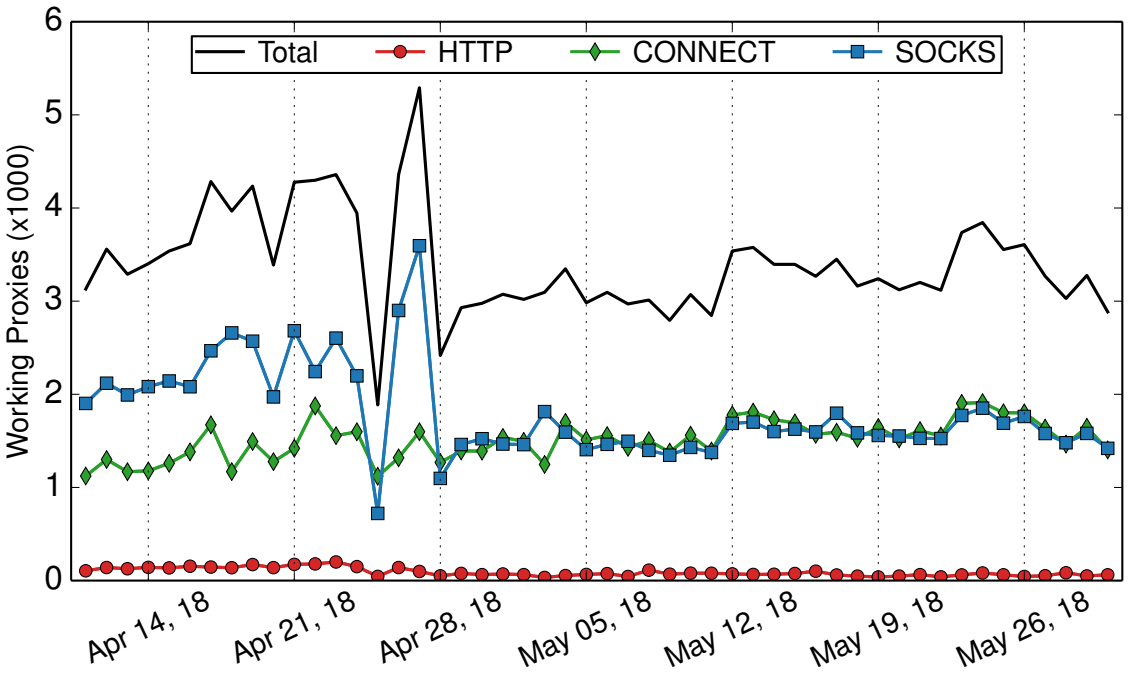

Figure 4.4: Working proxies, by type, over time.

4.3 Proxy Availability

The number of unique proxies listed by proxy aggregator sites, over time, is shown in Figure 4.3. The median number of proxies listed on the aggregator sites over all days in the measurement period is 41,520, with a range of [38,843; 48,296]. In total, during the course of our study, we indexed approximately 107,000 unique proxies that were listed on aggregator sites.

We find that more than 92% of open proxies that are advertised on proxy aggregator sites are offline or otherwise unavailable. Figure 4.4 plots the number of *responsive* proxies over time for which we were able to establish at least one connection and retrieve content. We remark that Figure 4.4 includes proxies that returned unexpected content. Across our measurement study, the median daily number of responsive proxies is 3,283; the medians for HTTP, CONNECT, and SOCKS proxies are 1,613; 1,525; and 74, respectively.

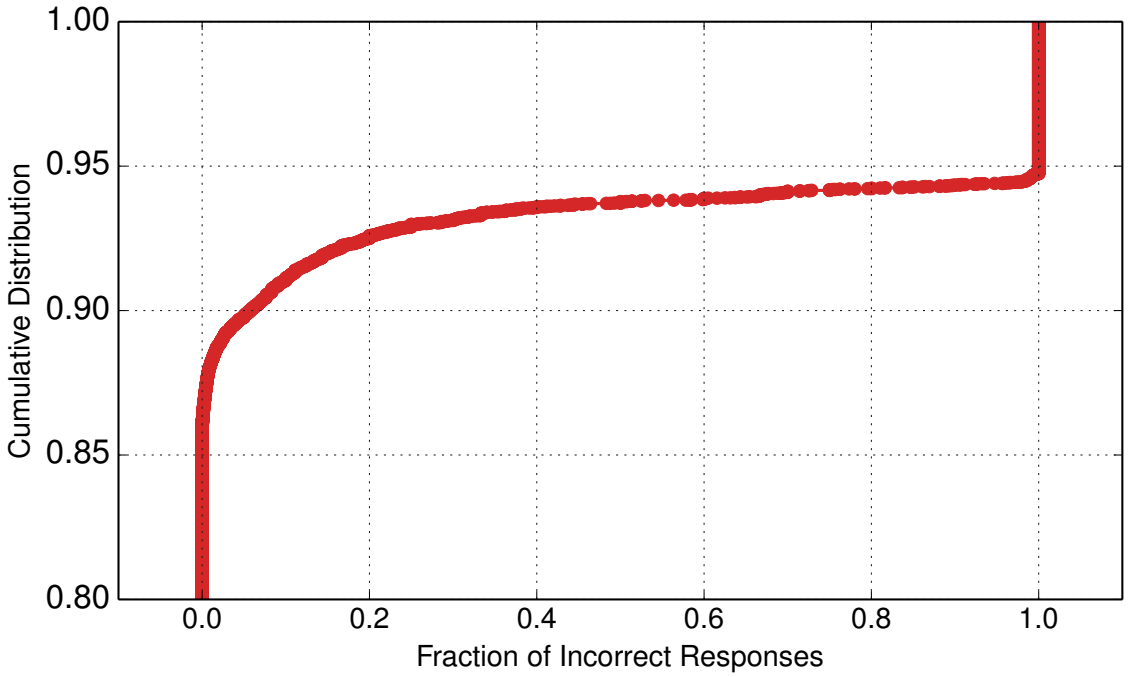

Figure 4.5: Cumulative distribution of the proxies' failure rate. Fraction of responses that constituted unexpected content.

4.3.1 Expected vs. Unexpected Content

Proxies do not always return the expected content. However, not all unexpected content is malicious. For example, many tested proxies return a login page or a page conveying an authentication error, regardless of the requested URL. These indicate that the listed open proxy is either misconfigured, actually a private proxy, or is some other misconfigured service. Here, we answer the question: to what extent do listed open proxies return the expected content?

For our analysis, we consider the proxies that have responded to at least one proxy request with a non-zero byte response and a 2xx HTTP response code that indicates success (*i.e.*, in $[200, 299]$). For each such proxy, we determine its failure rate, which we define as $1 -$ success rate. That is, a proxy's failure rate is the fraction of returned responses that constitute unexpected content.

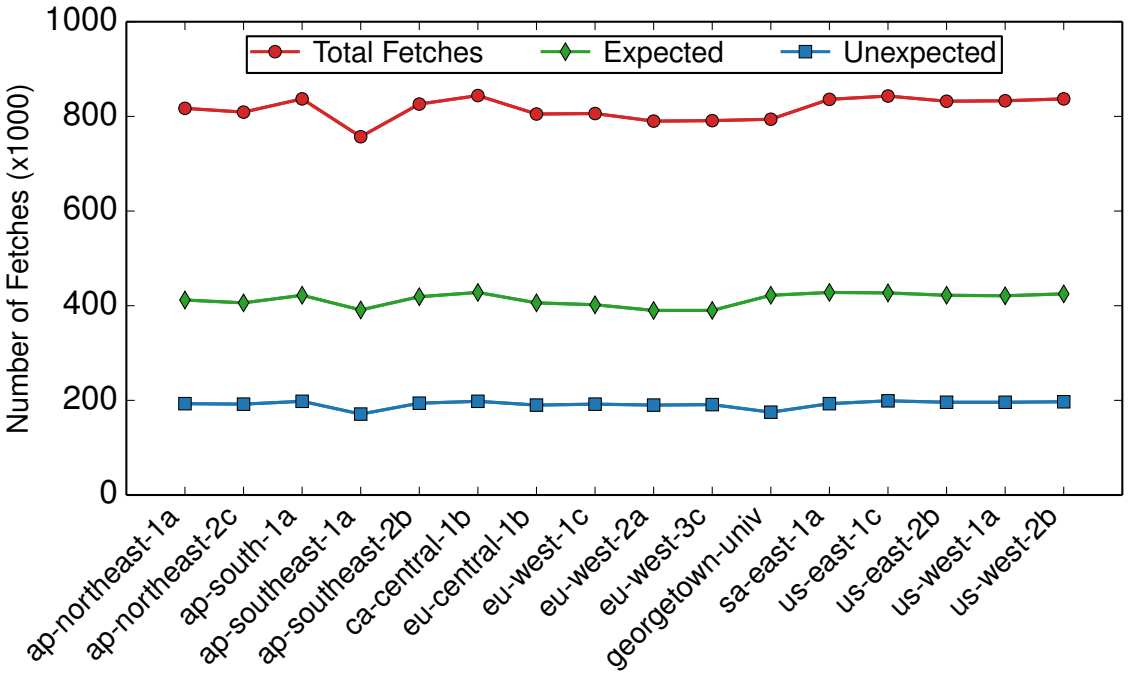

Figure 4.6: Fraction of fetches, by client location. Total fetches and fetches with expected and unexpected content, by client location.

Figure 4.5 shows the cumulative distribution (y-axis) of these proxies' failure rates (x-axis). Of proxies that respond to proxy requests with 2xx HTTP success codes, we find that 92.0% consistently deliver the expected content. Alarmingly, approximately 8% of the proxies at least sometimes provided unexpected content, and 3.6% of the proxies *consistently* returned unexpected content – all with HTTP response codes that indicate success. In §4.4, we explore cases in which the content has been purposefully and maliciously manipulated – for example, to insert spurious ads in retrieved HTML content. Further, as shown in Figure 4.6, the behavior of proxies – that is, whether they returned expected or unexpected content – does not significantly vary with the location of the requesting client.

4.3.2 Anonymity

Open proxies are sometimes used to provide weak form of sender anonymity by hiding a user's IP address. We find that such a strategy is mostly ineffective, with nearly two-thirds of tested proxies exposing the requestor's IP address.

In more detail, we inspect the HTTP *request* headers that are sent by a proxy to the destination webserver. These include the headers added by our client toolchain (specifically: `User-Agent` and `Accept`) and those inserted by the proxy. To accomplish this, we constructed and hosted a simple web application that records HTTP request headers. We then examined the request headers that were produced when we used a client at Georgetown University to access our web application through each proxy. Of the proxies that were able to connect to our web application (13,740), we found that 66.08% (9,079) inserted at least one header (most commonly, `X-Forwarded-For`) that contained the IP address of our client.

4.4 HTML Manipulation

We begin our study of malicious behavior among open proxies by considering the manipulation of fetched HTML content. In the absence of end-to-end SSL/TLS (*i.e.*, the use of HTTPS), a malicious proxy can trivially either modify the web server's response or respond with arbitrary content of its choosing.

To detect such misbehavior, we fetch an HTML page via the open proxies each day between 2018-05-04 and 2018-05-31. For simplicity, we focus on the HTML pages fetched through the client at Georgetown University. We observe a median of 1,133 successful requests (*i.e.*, completed requests with `2xx` HTTP response code) per day.

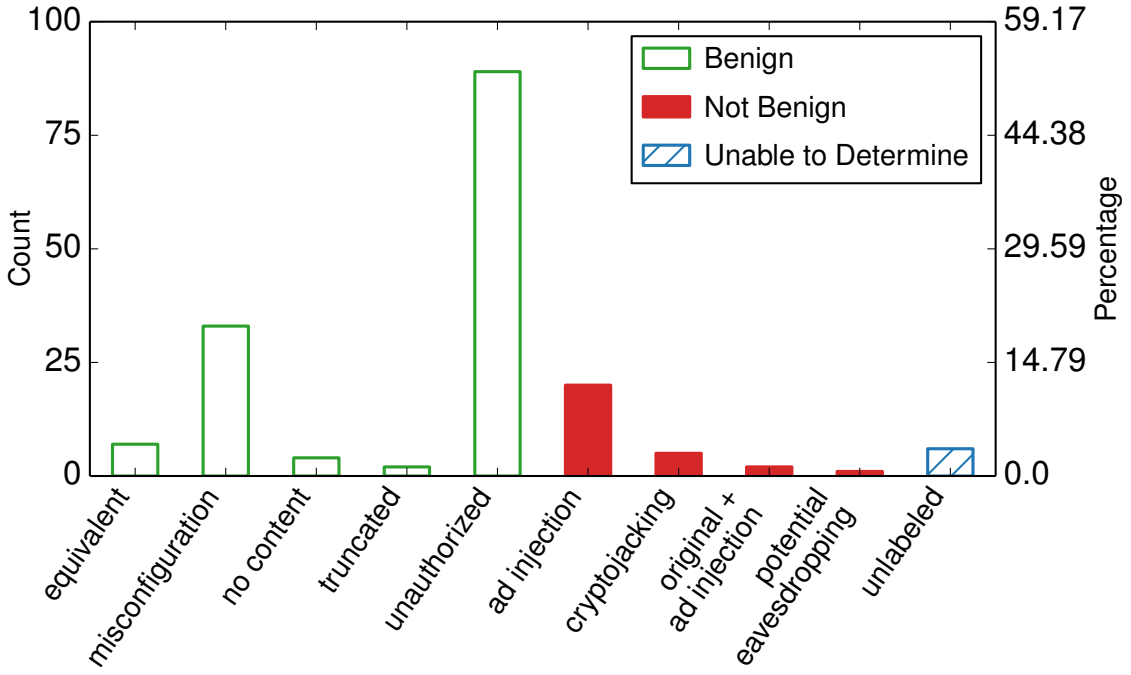

Figure 4.7: Classification of modified HTML retrieved on 2018-05-07.

Overall, we find that 10.5% of requests for HTML pages produced *un*expected HTML content. As noted above, not all unexpected content necessarily corresponds to malicious behavior. Since the observed fraction of unexpected HTML content is almost the same for each day of our measurement study, we perform an in-depth analysis for a random day, 2018-05-07, to determine whether the HTML manipulation could be categorized as malicious. On that date, we observe requests through 1,259 proxies, of which 169 (about 13.4%) return unexpected HTML content. (We discard requests that did not yield 2xx success HTTP response codes.)

We manually analyze these 169 *modified* HTML files. Our methodology is as follows. First, we inspect each file and divide them into two categories: benign (non-malicious) files and suspicious files. The latter category represents HTML files that contain Javascript code (either directly in the file or through the inclusion of an external Javascript file).

We then collectively inspect each of these benign and suspicious files and classify them into five benign classes and four non-benign classes. This inspection process involves manually examining the files, rendering them in a browser on a virtual machine, and potentially visiting Javascript URLs that were inserted. The five classes that we posit do *not* indicate maliciousness are as follows:

- *Equivalent:* the fetched HTML renders equivalently in a web browser to the expected content. Oddly, we found instances in which HTML tag attributes were inconsequentially reordered; for instance, tags such as `<IMG SRC="A" ALT="B">` are rewritten as `<IMG ALT="B" SRC="A">`.
- *Misconfiguration:* the retrieved content constituted error pages, often displaying "not accessible" messages.
- *No Content:* the page contained no content.
- *Truncated:* the retrieved page was a truncated version of the expected content.
- *Unauthorized:* the pages showed error messages such as "invalid user" or "access denied."

The identified classes of proxy misbehavior are:

- *Ad Injection:* the proxy replaced HTML content with Javascript that rendered extraneous advertisements.
- *Original + Ad Injection:* the returned HTML contained the expected content, but also included a single ad injection.
- *Cryptojacking:* the returned HTML include Javascript code that would cause the user's browser to mine cryptocurrencies on behalf of the proxy.

```
<script src="https://www.hashing.win/scripts/min.js">
</script>
<script>
    var miner = new Client.Anonymous(
        '[redacted]',
        { throttle: 0.1 }
        );
    miner.start();
</script>
```

Figure 4.8: Cryptojacking Javascript injected by open proxies. Injected HTML code with link to remote cryptojacking Javascript. The user identifier has been redacted.

- *Potential Eavesdropping:* the retrieved page contained Javascript that caused the user's browser to visit pages from its history, potentially revealing these pages to the proxy if they were previously visited without the use of the proxy.

The classifications of the modified HTML files with their counts and percentages are shown in Figure 4.7. We were unable to classify six HTML responses, described in the Figure as "Unlabeled."

Overall, we find that approximately 80% of the unexpected HTML responses are benign. Approximately 72% of the pages contained errors, likely due to private access proxies or due to misconfigured proxy software.

We find that 16.6% of unexpected HTML responses (and 2.2% of overall proxy requests for HTML content) correspond to malicious activity, including ad injection, cryptojacking, and eavesdropping. Among the malicious activity, most prominent (13%) is ad injection.

Perhaps most interestingly, we find that about 3% of the files include Javascript that performs cryptojacking – that is, the unauthorized use of the proxy user's processor to mine cryptocurrencies in the background. Upon further inspection, we determined that each such instance uses the same injected Javascript code, shown in Figure 4.8.

The referenced `min.js` script is obfuscated Javascript. After decoding it, we determined that it is similar to Coinhive's [16] Monero cryptocurrency [17] (advertised as a "secure, private, and untraceable" cryptocurrency) mining Javascript. The 64 character long argument to the Client constructor serves as the identifier for the user to be paid in the original Coinhive setting. (We redact this parameter in Figure 4.8 because it potentially identifies a criminal actor.) Finally, we note that all Coinhive endpoints described in the copied Coinhive script are replaced with other domains, indicating that whoever is running the infrastructure to collect the mining results is not using Coinhive's service.

HTTP Header Manipulation. We found a few instances of HTTP request header manipulation by proxies. Seven proxies changed the User-Agent HTTP request header before forwarding. Two of these proxies set it to libcurl-agent/1.0, one to ConBot, one to BonveioAbitona/2.1.0. Out of these seven proxies, two of them also manipulated the "Accept" HTTP request header to include additional types. All of these seven proxies always returned unexpected content for all requests. We conjecture that these header manipulations are likely due to the (mis)configuration of proxy softwares used by these proxies.

4.5 SSL/TLS Analysis

We find that 70% (14,607/20,893) of the proxies that fetch expected content at least once allow TLS traffic to pass through them.

We emphasize that supporting HTTPS incurs no overhead on behalf of the proxy, since the proxy is not involved in the (end-to-end) cryptography and is merely transporting the ciphertext. It is worth noting that as of early June 2018, more than 70% of loaded web pages were retrieved using HTTPS [97]. The lack of universal HTTPS support among the open proxies effectively forces some users to downgrade their security, bucking the trend of moving towards a more secure web.

We consider misbehavior among two dimensions: attempts at decreasing the security of the communication through TLS MitM or downgrade attacks, and alteration of the content.

SSL/TLS Stripping. We first examine whether any of the proxies rewrite HTML `<A>` link tags to downgrade the transport from HTTPS to HTTP. For example, a malicious proxy could attempt to increase its ability to eavesdrop and modify unencrypted communication by replacing all links to `https://example.com` with `http://example.com` on all webpages retrieved over HTTP. To perform this test, we fetched via each proxy a HTML page hosted on a web server at Georgetown University over HTTP. This HTML page contained six HTTPS links. We did not find evidence of proxies stripping SSL by replacing the included links.

SSL Certificate Manipulation. In this experiment, we search for instances of SSL/TLS certificate manipulation. While interfering with an HTTPS connection would cause browser warning messages, numerous studies have shown that even security-conscious users regularly ignore such warnings [25, 128].

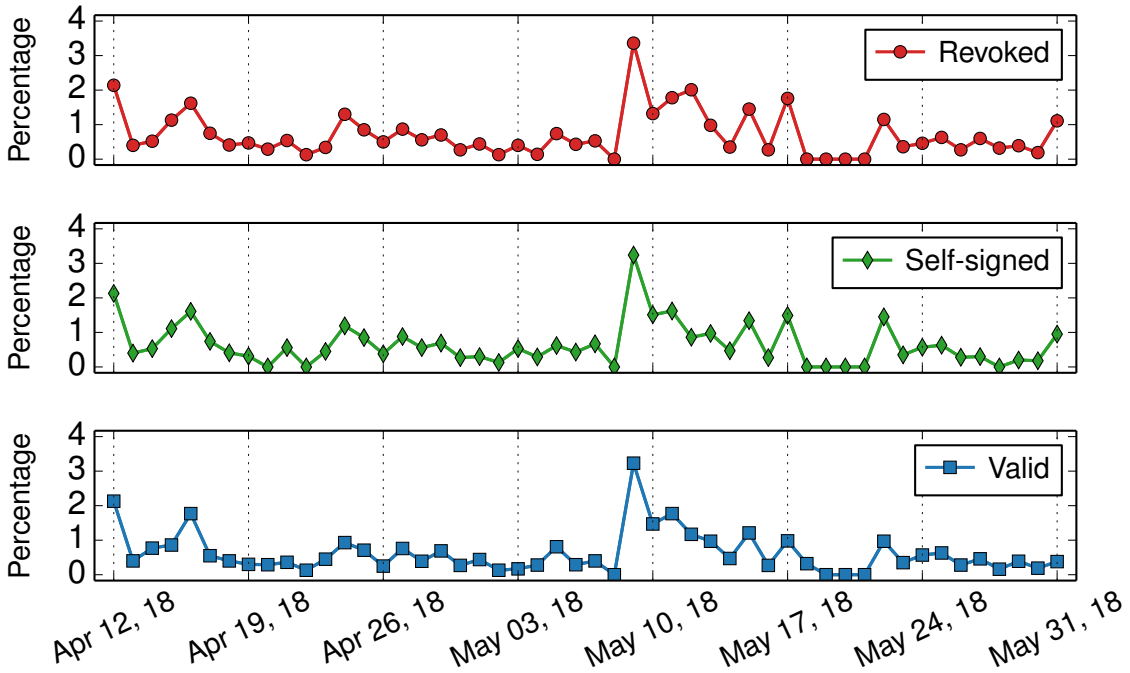

Figure 4.9: Percentage of eavesdropping open proxies. Percentage of proxies that return expected content but perform TLS/SSL MitM, for websites that use revoked (*top*), self-signed (*middle*), and valid and unexpired certificates (*bottom*).

We test the proxies against two categories of domains: one with a valid and verifiable SSL certificate hosted on a web server at Georgetown University; and domains with incorrect or invalid SSL certificates (`https://revoked.badssl.com/`, `https://self-signed.badssl.com/`). We include the latter category since we posit that a smart attacker might perform SSL MitM only in instances in which the connection would otherwise use revoked or self-signed certificates; in such cases, the browser would issue a warning even in the absence of the proxy's manipulation.

To detect SSL/TLS certificate manipulation, we fetch the three domains (listed above) via the open proxies each day between 2018-04-12 and 2018-05-31. For simplicity, we focus on the pages fetched through the client at Georgetown University.

Overall, we find that 1.6% (1,524/93,417) of proxies that support HTTPS perform TLS/SSL MitM by inserting a modified certificate. To determine whether

any of the proxies performing SSL/TLS MitM might belong to a known botnet, we searched the SSL Fingerprint Blacklist and Dyre SSL Fingerprint Blacklist [21] for the modified certificates' fingerprints. We did not find any blacklisted certificates.

We next consider proxies that fetch the *expected content* but modify the SSL/TLS certificate. Such behavior suggests that the proxy is eavesdropping on HTTPS connections. The percentage of such eavesdropping proxies, per day, for the different categories of certificates is plotted in Figure 4.9. Overall, 0.85% of the proxies that return expected responses appear to be eavesdropping. We did not find any evidence of proxies selectively targeting incorrect or invalid SSL certificates.

Finally, we analyze the modified TLS certificates inserted by the eavesdropping proxies when the requested site is `https://revoked.badssl.com/` or `https://self-signed.badssl.com/` (*i.e.*, when the genuine certificate is revoked or self-signed, respectively). We find that there were 435 modified certificates from 21 unique issuers. The issuer common name (CN) strongly suggests that 19% (4/21) of the issuers were schools. We posit that the proxies that modified these certificates were operated by schools and were incorrectly configured to serve requests from any network location. Interestingly, all of the certificates inserted by these school proxies had the expected subject common name and were valid (but not normally verifiable, since the school is not a root CA). This leads to an interesting result: if the schools pre-installed root CA certificates on students' or employees' computers, then they are significantly degrading the security of their users. That is, they are masking the fact that requested webpages have expired or revoked certificates by replacing these invalid certificates with ones that would be verified. This is similar to the effects observed by Durumeric et al. [56] in their examination of enterprise middleware boxes, which also degraded security by replacing invalid certificates with valid ones signed by the enterprise's CA.

HTML Manipulation. Performing TLS MitM also allows a malicious proxy to modify the content of the response. To detect such behavior, we fetch an HTML page over SSL/TLS via the open proxies each day between 2018-05-22 and 2018-05-31. For simplicity, we focus on the HTML pages fetched through the client at our local institution. We observe 5,700 successful requests (*i.e.*, completed requests with a 2xx HTTP response code) per day, of which about 0.84% (48/5700) had modified the SSL/TLS certificate. Of the 48 modified responses, 39.6% (19/48) had an HTML MIME-type. Interestingly, approximately 68% (19/28) of the CONNECT proxies that inject false certificates also modify HTML content.

We manually analyzed these 19 *modified* HTML files using the same approach as described in §4.4. We find that all 19 instances corresponded either to "access denied" pages or otherwise indicated a proxy misconfiguration. We did not find any malicious activity.

4.6 Comparison with Tor

We now conduct a similar measurement study of Tor [53], the most widely used anonymity network. Recall from §3.1 that Tor provides anonymous TCP communication by routing user traffic through multiple relays (typically three) using layered encryption. The first relay in the path (or circuit) is the guard relay, and the final relay through which traffic exits is the exit relay. The original data transmitted is visible only at the exit relays. Therefore, unless e2e encryption is used, data can be eavesdropped by malicious or compromised exits. Prior research studies have found evidence of malicious behavior such as interception of credentials, *etc.*, by a small fraction of Tor exit relays, especially when the traffic was not e2e

encrypted [38, 105, 139]. We now compare the level of manipulation as observed using open proxies to that when content is fetched over Tor.

We modify Exitmap [139], a fast scanner that fetches files through all Tor exit relays. To maintain consistency with our earlier experiments, we fetched a HTML page as described in §4.4 over HTTP, and accessed the same HTTPS URLs as described in §4.5. We fetched these files each day through every available Tor exit relay between 2018-05-06 and 2018-05-31, during which the median number of available exit relays was 722.

Approximately 13.8% of connections and 1.8% of fetches timed out when using Exitmap. This is unsurprising since Exitmap does not perform the same (and necessary) bandwidth-weighted relay selection as the standard Tor client [53, 139].

Over our 26 day Tor experiment, we found no instance in which a Tor exit relay manipulated either file contents or SSL/TLS certificates. Comparing our results to §4.4-4.5, this strongly suggests that Tor is a more trustworthy network for retrieving forwarded content. However, since Tor exits may still passively eavesdrop (which we would not detect), we concur with the conventional wisdom that e2e encryption (*i.e.*, HTTPS) is appropriate when using Tor.

We evaluate the performance of proxies by considering *goodput*, computed as the size of a fetched file (we use a 1MiB file) divided by the time taken to download it through a proxy. We consider only instances in which the response reflects the expected content (*i.e.*, the 1MiB file). We also compare the performance of Tor to that of the open proxies. For Tor, we rely on data from the Tor Metrics Portal [130]; specifically, we use the median time taken per day to download a static file of size 1 MiB and derive Tor's median throughput between 2018-05-06 and 2018-05-31. Figure 4.10 shows the Tor and open proxies' median throughput per day. We find

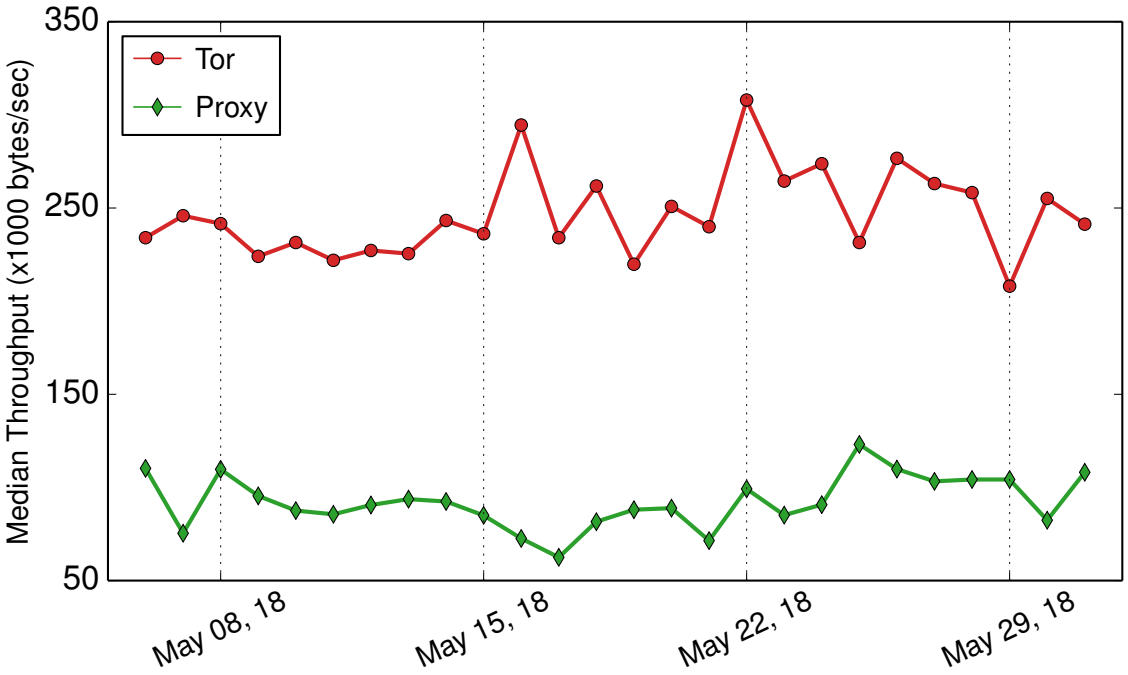

Figure 4.10: Tor vs. Open proxies median throughput. Tor vs. open proxies median throughput per day while fetching a static file of size 1 MiB.

that the Tor median throughput is roughly twice the open proxy throughput. This suggests that Tor performs relatively faster than the open proxies.

4.7 Summary

In this chapter, we performed an extensive evaluation of the Internet's open proxies, that are often used to achieve a limited degree of sender anonymity. We found numerous instances of misbehavior, including the insertion of spurious ads and cryptocurrency-mining Javascript, and TLS MitM. Moreover, we found that 92% of advertised proxies listed on open proxy aggregator sites are nonfunctional. In contrast, our nearly monthlong study of the Tor anonymity network found zero instances of misbehavior and far greater stability and goodput, indicating that Tor offers a safer and more reliable form of proxied communication.

In the next chapters, we will explore the much more complicated and challenging problem of measuring the usage of anonymity networks.

Chapter 5

Private and Robust Statistics Collection for Tor

In this chapter, we introduce HisTorϵ, a differentially private statistics gathering system that can be used to conduct accurate measurement studies on anonymity networks (primarily Tor) without endangering the network's users. HisTorϵ is scalable, incurs low overheads, and unlike prior research [62, 80], provides strong integrity guarantees and is robust against adversarial manipulation.

HisTorϵ's robustness is mostly achieved by forgoing general counting queries (as supported by PrivEx or PrivCount) in favor of supporting only *binning* queries. A useful example of the type of queries that HisTorϵ supports is a histogram. For instance, consider a query that aggregates the number of client connections observed by guards. In case of counting queries supported by PrivEx [62] and PrivCount [80], each data collector (*i.e.*, entry guard) reports the number of client connections observed by it, and the system computes the sum of those values. So, a malicious entry guard can drive the client connection count towards an arbitrary value of its choosing, without risking detection (shown in solid red in the left side of Figure 5.1). By contrast in HisTorϵ, each entry guard can only contribute "0" or "1" to the total count in a bin. HisTorϵ uses Goldwasser-Micali (GM) encryption [75] to ensure that encrypted values are either treated as 0 or 1. Therefore, the influence of a ma-

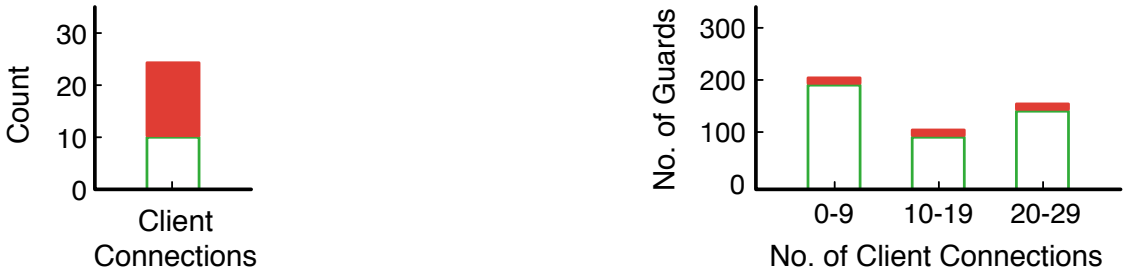

Figure 5.1: Influence of a malicious relay on counting vs. binning queries. The influence of a single malicious relay on the aggregate result of a counting query (*left*), and a histogram query (*right*). The value reported by the malicious relay is shown in solid red.

licious data collector (for example, entry guard) over the total aggregate is strictly bounded *i.e.*, it can contribute at most 1 to each bin as shown in solid red in the right side of Figure 5.1.

Like PrivEx and PrivCount, HisTorϵ also provides resistance to the so-called "compulsion attack" in which a relay operator can be compelled to release sensitive information through a subpoena or threat of violence. Gathering statistics in an anonymity network (such as Tor), poses privacy risk since such sensitive statistics would not otherwise be collected at the relays. For example, Tor does not currently log client connections to guards; queries which ask for a histogram of the number of guard connections thus may impose additional privacy risks. We design HisTorϵ to mitigate such risks through the use of encrypted data structures we call oblivious bin counters. Simply put, we use efficient cryptographic techniques and data structures to maintain counters that the relay operators cannot read on their own.

5.1 Manipulating PrivEx

PrivEx [62] is the first system for private data collection on the live Tor network. In comparison with HisTorϵ, which we introduce in the next section, PrivEx has

a similar system model, but differs in that relays individually contribute their own value, and the system computes the sum of those values. That is, PrivEx supports counting queries, and there is no protection against an attack on the integrity of the statistics. This enables a malicious contributor (*e.g.*, a relay) to submit an arbitrary value that can significantly impact the aggregate result. Elahi et al. posit that range proofs could potentially be added to PrivEx to bound the impact of untrue statistics. However, as the authors admit, such techniques lead to significant computation and communication overheads, and does not remove the threat entirely.

We demonstrate this type of manipulation by making a trivial modification to the PrivEx source code as described in §1.3. We use the same default parameters as included in the PrivEx source code, and consider a query that aggregates the number of visits destined to 15 particular websites. In our experiment, we modify the behavior of a single relay to falsely claim that it has observed 1000 visits to website #2. Figure 1.1 on page 8 shows the result of this manipulation. The actual distribution of website visits (shown in hollow green) indicates that website #6 is the most popular site visited via Tor. However, the modified (and noised) distribution reported by PrivEx (shown in solid red) paints a different picture: both website #2 and #6 appear to be popular amongst Tor users.

The above example is of course one of many possible manipulations. More generally, a single malicious relay that participates in PrivEx can supply any (false) count to manipulate the aggregated result, and do so without risking detection.

PrivCount [80] which extends PrivEx to make it suitable for a small-scale research deployment, is also vulnerable to this type of manipulation. However, we emphasize that the above "attack" falls entirely outside of the threat models considered by PrivEx and PrivCount. That is, we are not disclosing errors in their design or implementation. We simply argue that though PrivEx and PrivCount raise the

bar for safe Tor measurements, their lack of integrity guarantees can be problematic as Tor is a volunteer-operated network. Indeed, prior research has found evidence of malicious behavior [139] by relay operators. Therefore, privacy should be combined with integrity protections to provide a more useful statistics gathering service.

5.2 Overview

HisTorϵ's goal is to provide differentially private data collection for Tor that is robust against adversarial manipulation. We now describe the system model (§5.2.1), threat model (§5.2.2), query capabilities (§5.2.3), and operation (§5.2.4) of HisTorϵ.

5.2.1 System Model

There are three types of participants in HisTorϵ:

Data collectors (DCs) [62] are relays that collect statistics that will later be aggregated. Example of DCs and the data that they collect include guard relays that count the number of client connections, middle relays that count requests to Tor Hidden Service introduction points, and exit relays that keep track of exit bandwidth. Our aim is to provide a secure and private statistics gathering technique for Tor that is sufficiently general to be adapted for many such queries.

As with vanilla Tor, HisTorϵ relies on secure communication via TLS, and we use Tor's existing directory infrastructure as a trust anchor to locate public keys and authenticate messages. We assume that the DCs involved in a query are publicly known. Since relays are already heavily burdened in Tor, we aim to keep both the computation and communication overheads of HisTorϵ low for the DCs.

The *analyst* is a party that issues queries to the data collectors and receives noised query responses. We envision that, at least initially, the analyst will be the maintainers of Tor.

Finally, we introduce three *mixes* that are third parties that provide the required amount of differentially private noise. Mixes are dedicated servers responsible for enabling HisTorϵ queries. We assume that all parties can obtain the mixes' public keys, which for example, could be attested to by the Tor directory authorities.

Also included in the HisTorϵ ecosystem are the Tor users who use the Tor client software to communicate via Tor, the destinations that the users are visiting, and the Tor directory servers and mirrors. Since HisTorϵ imposes no changes to the normal operations of Tor, we mostly omit discussing these components unless they are pertinent to the security and privacy properties of HisTorϵ.

5.2.2 Threat Model and (Informal) Security Guarantees

We consider both internal and external threats to privacy and data integrity:

Internal Adversaries. DCs are volunteer-operated relays and can be malicious. A malicious DC is a Byzantine adversary that can, for example, disobey HisTorϵ protocols, submit false statistics, and/or refuse to participate. Malicious DCs may also collect and leak sensitive information (*e.g.*, the IP addresses of clients or destination addresses); such leakage is also possible with existing Tor relays, and we do not consider defenses against such behavior here. HisTorϵ ensures that no colluding group of DCs can reveal more information than would otherwise be available by pooling their knowledge.

Malicious DCs may attempt to manipulate query results by reporting erroneous data, as in §5.1. If c is the number of DCs that participate in a query and f is the fraction of the participating DCs that are malicious, then HisTorϵ guarantees

that the maximum influence over the aggregate result is $\pm fc$. This is a direct consequence of applying the (ϵ, δ)-differential privacy scheme of Chen et al. [42]: each relay (malicious or not) can contribute at most 1 to each element in its supplied vector. If a DC refuses to participate in a query or submits a malformed vector, its bit vector is considered to be all 0s. Consequently, malicious DCs cannot disrupt (*i.e.*, cause denial-of-service) HisTorϵ queries by submitting false or malformed data.

Malicious mixes may also behave arbitrarily. HisTorϵ provides strong privacy guarantees when (1) no more than one of the mixes is malicious and (2) a malicious mix does not collude with a malicious analyst. HisTorϵ employs secret sharing techniques to ensure that non-colluding mixes cannot learn un-noised query results.

Mixes can attempt to manipulate query results by adding false values, improperly constructing noise vectors, or modifying or discarding the encrypted vectors it receives from the DCs. HisTorϵ's integrity guarantees ensure that the analyst can detect manipulated query results as long as one of the mixes is honest.

In HisTorϵ, the analyst issues queries and receives noised results from the mixes. We consider a malicious analyst that colludes with other parties to attempt to learn non-noised answers to queries. HisTorϵ achieves (ϵ, δ)-differential privacy when no more than one mix is malicious and no malicious mix colludes with the analyst. If a malicious DC colludes with either a mix or the analyst, no information that is not already available to the malicious DC is revealed.

External Adversaries. HisTorϵ is robust against an external adversary that observes all HisTorϵ-related communication. All HisTorϵ exchanges are secured by TLS. We assume that public keys can be reliably retrieved – for example, by leveraging Tor's existing directory infrastructure – and that all properly signed messages can be authenticated.

We also consider an external adversary that applies pressure (*e.g.*, through a subpoena or threat of violence) to an honest DC to reveal its individual counters. We describe in §5.3 techniques that limit the amount of information that can be "handed over" by a pressured honest relay to an adversary.

5.2.3 Queries

In HisTorϵ, for each query, each DC i contributes a bit vector $\mathbf{v}_i = \{v_{i,1}, \ldots, v_{i,b}\}$, of length b, where b is a parameter of the query. For ease of exposition, we refer to each vector position $\{v_{i,j}\}$ (where $1 \leq j \leq b$), as a *bin*. We adopt the distributed (ϵ, δ)-differential privacy scheme of Chen et al. [42], described in §5.8. Suppose c DCs respond to the analyst's query. Then to provide privacy, the mixes collaboratively add in n encrypted random noise vectors, where n is calculated as in Eq. 3.2.

The $c + n$ DC and noise vectors are then shuffled by the mixes to provide indistinguishability, and returned to the analyst. Letting $d_{i,j}$ be the jth element of vector i (where $1 \leq i \leq c+n$), the analyst obtains the (noised) aggregate result $\mathbf{a}_j$ as

$$\mathbf{a}_j = \sum_{i=1}^{c+n} d_{i,j} - \frac{n}{2} \tag{5.1}$$

for all $1 \leq j \leq b$ (the number of elements in a bit vector). Essentially, the $-n/2$ term corrects for the expected amount of noise added by the mixes. We describe the details of the protocol in §5.2.4.

In their work, Chen et al. [42] set $\delta < 1/c$ in Eq. 3.2. However, Dwork et al. [59] prove that such a value of δ is very dangerous, as it permits compromising the privacy of "just a few" number of clients. Therefore, we set δ to a much lower value, on the order of $10^{-6}/u$, where u is the number of Tor users HisTorϵ will

protect. That is we limit the chance of a privacy"failure" affecting any u Tor users to 10^{-6}.

HisTorϵ supports two types of queries: class queries and histogram queries.

Class Queries. In a class query, each bin j is assigned a class label C_j. For Tor, potentially useful class labels include (but are not limited to) websites, protocols/ports, *etc.* seen by exit relays and client version numbers seen by guards. The semantics of class queries allow the analyst to ask for all $j \in [1, b]$: *how many relays have witnessed the event described by the label C_j?*

The analyst specifies the semantic meaning for each bin. For example, in the case of reporting which ports have been seen by exit relays, the analyst can specify a query such that the first bin in DCs' bit vectors indicates whether the exit has witnessed http traffic, the second bin indicates whether it has seen ssh traffic, *etc.* By examining the aggregated and noised results, the analyst learns approximately how many exit relays have seen the specified traffic types.

Histogram Queries. Histogram queries allow the analyst to learn the distribution of some value, taken over the DCs. As examples, histogram queries can inform the analyst of the distribution of client connections seen by guards or the bandwidth seen by exit relays.

For histogram queries, each DC maintains an encrypted counter of the relevant statistic (*e.g.*, observed bandwidth). Each bin j is assigned an interval $[L_j, U_j)$ where $L_j, U_j \in \mathbb{Z}$ such that $L_j \geq U_{j-1}$ when $j > 1$. When a bin $b_j = 1$, this indicates that the encrypted counter is in the range $[L_j, U_j)$. Note that at most one bin belonging to a DC is set to 1; all other bins are set to 0. (This is unlike a class query; there, multiple bins/classes can be set to 1.)

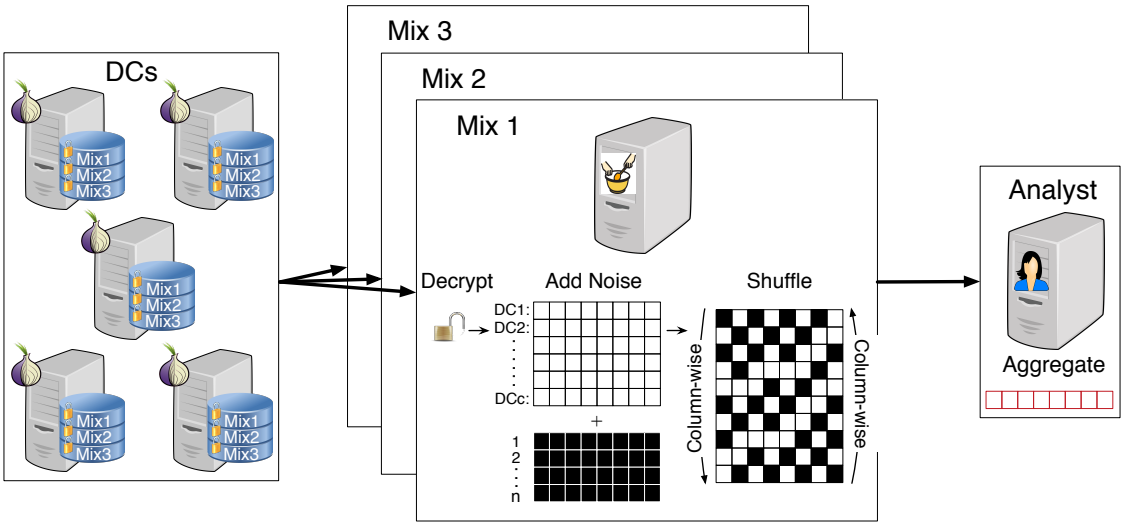

Figure 5.2: HisTorϵ protocol overview. An overview of HisTorϵ query processing. Each mix receives an encrypted vector from each DC. The mixes then add noise and perform a column-wise shuffle. The resulting vectors are then sent to the analyst, who in turn can compute the aggregate result.

The vector sum over all DCs' bin vectors (*i.e.*, $\sum \mathbf{v_i}$) yields the distribution of the DCs' counters. As explained next, mixes add random noise vectors to apply differential privacy to this distribution.

5.2.4 Operation

HisTorϵ operates in loosely synchronized hour-long epochs. At the beginning of each epoch, each DC zeroes all of its bins. As with PrivEx [62], HisTorϵ ensures that data sets do not carry over between queries; that is, no more than one query within an epoch can cover a given bin. We enforce independence between different epochs by zeroing all counters. (We include discussions of the absoluteness of this independence and the effect of a privacy budget for differentially private queries in §5.9.)

HisTorϵ's workflow begins when the analyst issues a query to the DCs and mixes.This is a manual process that requires configuring Tor relays to report the statistic of interest (*e.g.*, connection counts, observed bandwidth, *etc.*). We envision that future versions of HisTorϵ will support SQL-like semantics to automate this process. Our prototype implementation, described in more detail in §5.7, uses Tor's existing statistics module and can be easily configured to aggregate the data that Tor already collects. We have already added hooks for bandwidth and connection counting.

Figure 5.2 shows how queries are processed in HisTorϵ. DCs maintain three redundant copies of encrypted counters (encrypted binary vectors). Each copy is encrypted using the public key of one of the three mixes.

At the end of the epoch, these encrypted vectors are further obfuscated by xor'ing with a random binary vector R. Xor'ing with the random vector ensures that mixes cannot learn the plaintext of the DCs' binary vectors, even after decrypting with their private GM key. (Recall that GM is a homomorphic cryptosystem with respect to xor [75].) The DCs send one copy of its GM-encrypted and xor'd vector to each mix, as shown in Figure 5.2. Additionally, the DCs communicate secret shares of R across the three mixes.

Each mix then decrypts the GM encryption, yielding the original binary vectors xor'd with a random vector. Each such vector is added to a matrix; that is, this matrix contains the xor-encrypted vectors from the DCs. Mixes then add n randomly generated rows to the vector, where n is computed according to Eq. 3.2 (where, $\delta = 10^{-6}/u$). Finally, the vector is randomly shuffled columnwise. Both the addition of the n rows of noise and the shuffling is performed using cryptographically secure random seeds, which are shared amongst the mixes. Consequently, the three mixes add identical noise vectors and perform the identical shuffle.

Finally, the mixes communicate the resultant matrices to the analyst as well as the shuffled secret shares of the R random vector. The analyst then combines the shares to obtain the shuffled R, and uses it to decrypt both the shuffled data and noise records (which are indistinguishable) in the matrix. To obtain the aggregate, the analyst then subtracts the expected noise value according to Eq. 5.1.

5.3 Oblivious Counters

To mitigate compulsion attacks, HisTorϵ minimizes the amount of information that DCs need to maintain through the use of *oblivious counters*. In this section, we assume that the DCs are honest. A malicious DC (relay) need not use oblivious counters and can trivially leak sensitive statistics, regardless of whether HisTorϵ is used.

Each DC maintains three binary vectors of length b, where b is determined by the query. Each binary element of the first binary vector is GM-encrypted using the public key of the first mix; the second vector is GM-encrypted using the public key of the second mix, and so on. For ease of notation, we focus below on one such GM-encrypted vector which we denote as $\overset{e}{\mathbf{v}} = \langle E_+(v_1), \ldots, E_+(v_b)\rangle$, where $E_+(\cdot)$ denotes encryption using the public key belonging to the pertinent mix. The scheme is identical for the three encrypted binary vectors maintained by each DC.

5.3.1 Oblivious Class Counters

Class queries allow the analyst to discover how many DCs encountered an event (see §5.2.3). Recall that each v_j corresponds to a class label C_j, which for example could denote a particular protocol or type of event. When $v_j = 1$, we say that the event has been observed by the DC; and $v_j = 0$ indicates that the event was not witnessed.

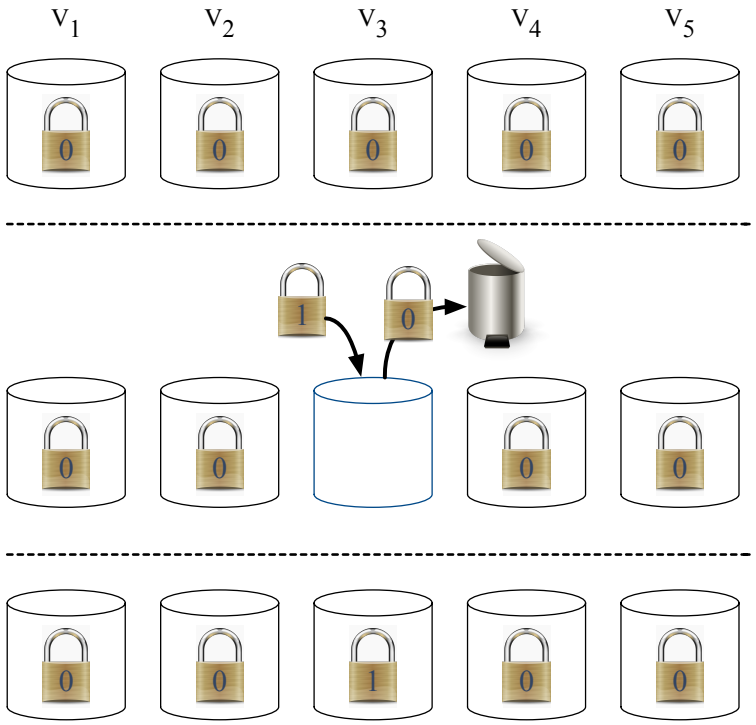

Figure 5.3: Maintaining an oblivious class counter. *Top:* The counters ($b = 5$) are initialized to GM-encryptions of 0. *Middle:* An event corresponding to class C_3 is observed, and $E_+(v_3)$ is replaced with $E_+(1)$. *Bottom:* The resulting encrypted binary vector.

At the beginning of each epoch, each DC zeroes its vector $\overset{e}{\mathbf{v}}$ by GM encrypting the bit 0, b times. Since the GM cryptosystem is a probablistic cryptosystem [75], we have that $\forall x, y \in [1, b]$, $x \neq y \Rightarrow E_+(v_x) \neq E_+(v_y)$ with high probability. That is, without knowledge of the corresponding private key, GM encryption guarantees that an adversary cannot determine whether $v_x \overset{?}{=} v_y$ when $x \neq y$.

When a DC observes an event corresponding to a class label C_j, it replaces the value of $E_+(v_j)$ in $\overset{e}{\mathbf{v}}$ with a new GM encryption of the bit 1. Note that if v_j was already 1, then the operation effectively replaces the old encryption of 1 with a new encryption of 1. This process is depicted in Figure 5.3.

Crucially, DCs do not store the plaintext vector elements $v_1, \ldots, v_b$, and instead maintain only the encrypted binary values $E_+(v_1), \ldots, E_+(v_b)$. Since the DCs also do not have the private key for decrypting the elements of $\overset{e}{\mathbf{v}}$, it trivially holds that an honest DC that does not know either v_j or the mix's private key cannot provide v_j to an adversary. That is, the above scheme trivially achieves resistance to the compulsion attack.

5.3.2 Oblivious Histogram Counters

In the case of histogram queries, recall that the analyst's query maps each vector element v_j to a range $[L_j, U_j)$ such that $L_j \geq U_{j-1}$ when $j > 1$. When v_j is 1, this indicates that the statistic of interest as measured by the DC is in the range $[L_j, U_j)$. Hence, at most one v_j is set to 1. As a special case, we set $U_b = \infty$.

Let w_j be the *bin width* of vector element v_j; *i.e.*, $w_j = U_j - L_j$. We do not require that $\forall x, y \in [1, b], w_x = w_y$, but we denote this special case as *equal width* bins.

As explained below, our oblivious histogram counter scheme requires equal width bins. Since we do not require that $w_x = w_y$ for all $x, y \in [1, b]$ – that is, the histogram query need not use equal width bins – we use an auxiliary binary vector with equal width bins, where the bin width is set to the greatest common divisor (GCD) of $w_1, \ldots, w_b$. More formally, we define $\overset{e}{\boldsymbol{\nu}} = \langle E_+(\nu_1), \ldots, E_+(\nu_\beta) \rangle$, where each element ν_j covers the range $[(j-1)g, jg)$ and g is the GCD of $w_1, \ldots, w_b$. As a special case, ν_β covers the range $[(\beta - 1)g, \infty)$. In practice, to reduce the size of $\overset{e}{\boldsymbol{\nu}}$, we carefully choose bin widths such that $\beta < 15000$. In the special case that the histogram query specifies equal width bins, we have that $b = \beta$ and $\overset{e}{\mathbf{v}} = \overset{e}{\boldsymbol{\nu}}$.

```
1 t ← t + 1
2 if t = g                                          // g is GCD and bin width
3  then
4     tmp ← E+(ν_{β−1}) ⊕ E+(ν_β)
5     ν⃗^e ← ν⃗^e ≫ 1                                  // Right shift, no wrap
6     E+(ν_β) ← tmp                                 // GM is xor homomorphic
7     E+(ν_1) ← E+(0)
8     t ← 0
9 end
```

Procedure IncrHistCounter

At the beginning of each epoch, each DC initializes its encrypted vector as $\vec{\boldsymbol{\nu}}^e = \langle E_+(1), E_+(0), \ldots, E_+(0)\rangle$ (all but the first element are encryptions of 0). It also maintains a counter t, initialized to 0.

When the DC observes the statistic of interest (number of connections, a KB of consumed bandwidth, *etc.*), it executes the IncrHistCounter procedure.

The IncrHistCounter procedure works by shifting the position of the encrypted 1 in the $\vec{\boldsymbol{\nu}}^e$ vector whenever the counter reaches the bin width. Line 6 handles the special "overflow" case: when the last bin is set to a 1, it always retains that 1 (recall that the last bin represents the range $[(\beta-1)g, \infty)$). An example invocation of IncrHistCounter is shown in Figure 5.4.

IncrHistCounter can be easily adapted for statistics that increase in increments greater than 1 (*e.g.*, observed bandwidth). In particular, if k is the increase in the statistic, then $\vec{\boldsymbol{\nu}}^e$ is right shifted $\lfloor (t+k)/g \rfloor$ positions and t is reset to $(t+k)$ mod g.

Since an honest DC does not maintain the plaintext values of $\nu_1, \ldots, \nu_\beta$ and does not know the decryption key, it cannot reveal which bin contains the 1, even if compelled to do so. Importantly, unlike oblivious class counters, oblivious histogram counters leak information – in particular, the counter t. More formally, we

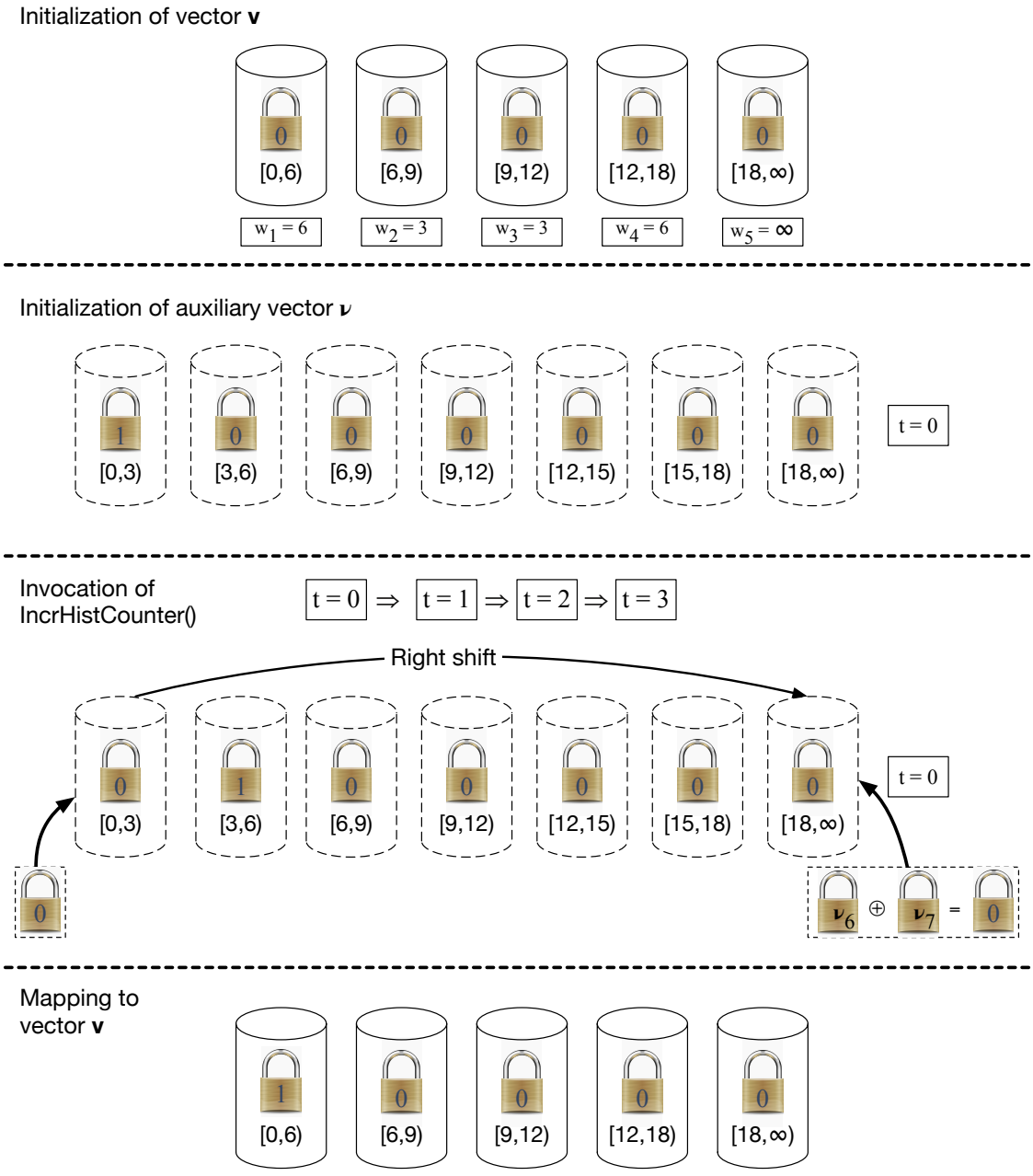

Figure 5.4: Maintaining an oblivious histogram counter. *Top:* The analyst defines bin widths for $\vec{\mathbf{v}}^{e}$. The GCD (g) is 3. *Second from top:* The initialization of $\vec{\boldsymbol{\nu}}^{e}$. *Second from bottom:* After three observations ($t = 3$), the bins in $\vec{\boldsymbol{\nu}}^{e}$ are right shifted. *Bottom:* At the end of the epoch, the values of $\vec{\boldsymbol{\nu}}^{e}$ are mapped to $\vec{\mathbf{v}}^{e}$.

leak $\lfloor \log_2 g \rfloor + 1$ least significant bits of the DC's measured value, $(\sum_{v_i \in \mathbf{v}} v_i \cdot L_i) + t$, when a compulsion attack takes place (since $t \in [0, g)$).

At the end of the epoch, the encrypted values of $\overset{e}{\boldsymbol{\nu}}$ need to be mapped back to $\overset{e}{\mathbf{v}}$. Since the width of the ranges covered by $\overset{e}{\boldsymbol{\nu}}$ are defined by the GCD of the ranges covered by $\overset{e}{\mathbf{v}}$, it holds that each $E_+(\nu_j) \in \overset{e}{\boldsymbol{\nu}}$ maps to a single $E_+(v_k) \in \overset{e}{\mathbf{v}}$. For example, in Figure 5.4, we have $E_+(\nu_1) \mapsto E_+(v_1)$, $E_+(\nu_2) \mapsto E_+(v_1)$, $E_+(\nu_3) \mapsto E_+(v_2)$, $E_+(\nu_4) \mapsto E_+(v_3)$, $E_+(\nu_5) \mapsto E_+(v_4)$, $E_+(\nu_6) \mapsto E_+(v_4)$, and $E_+(\nu_7) \mapsto E_+(v_5)$, where $\mapsto$ signifies the mapping between $\overset{e}{\boldsymbol{\nu}}$ and $\overset{e}{\mathbf{v}}$. Let $\mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})$ denote the set of elements of $\overset{e}{\boldsymbol{\nu}}$ that map to a given $v_k \in \overset{e}{\mathbf{v}}$. We can therefore compute

$$E_+(v_k) = \bigoplus_{E_+(\nu_j) \in \mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})} E_+(\nu_j) \tag{5.2}$$

Eq. 5.2 holds since at most at one element in $\mathcal{M}(v_k, \overset{e}{\boldsymbol{\nu}})$ is 1 and GM is homomorphic with respect to xor ($\oplus$).

5.4 Robust Differential Privacy

In this section, we describe how the DCs, mixes, and analyst interoperate to provide robust differential privacy. We explain how each DC tallies the statistic of interest using oblivious counters, and present the details of our HisTorϵ protocol for aggregating these statistics into a differentially private aggregate.

Let Mix_1, Mix_2, and Mix_3 be the three mixes. Mix_1 is referred as the master mix and the other two mixes (Mix_2 and Mix_3) are referred to as the slave mixes. All communication is assumed to be through secure TLS channels. The process of aggregating the individual DC counters is as follows:

Parameter Initialization. At the beginning of every epoch, Mix_1 generates five cryptographically secure random seeds: the common shuffling seed s, the common

random vector seeds p and q, and the pairwise mix seeds x_2 and x_3. It transmits $\langle x_3, p, q, s\rangle$ to Mix_2 and $\langle x_2, p, q, s\rangle$ to Mix_3. Then, Mix_2 generates a cryptographically secure random pairwise mix seed x_1 and transmits $\langle x_1\rangle$ to Mix_3.

Query Initialization. The analyst formulates a query, and transmits it to the master mix Mix_1. It specifies the privacy parameter ϵ and, in the case of a histogram query, a set of b bins $\langle [L_1, U_1), \ldots, [L_b, U_b)\rangle$. We discuss the practical aspects of selecting an appropriate value for ϵ in §5.9.

Query Forwarding. The master mix Mix_1 forwards the query (received from analyst) to the DCs. The master mix maintains a list, L_c, of the DCs that acknowledged the request.

DC Statistics Gathering and Response. For each query, each data collector D collects statistics during the course of the epoch using three sets of oblivious counters (see §5.3). There is one set of oblivious counters for each of the three mixes.

At the conclusion of the epoch, the DC performs the following operations:

(i) D maintains a series of encrypted oblivious counters $\overset{e}{\mathbf{v}}_i = \langle E_+(v_{i,1}), \ldots, E_+(v_{i,b})\rangle$ for $1 \leq i \leq 3$, where each element of $\overset{e}{\mathbf{v}}_i$ is encrypted with Mix_i's GM public key. To ease notation, we refer to the non-encrypted bit vector $v_{i,1}, \ldots, v_{i,b}$ as M.

(ii) D chooses b-bit random binary vectors $R, R_1, R_2, R_3 \in_r \{0,1\}^b$, where $\in_r$ denotes uniformly random selection.

(iii) D computes $R'_i = R \oplus R_i$, $1 \leq i \leq 3$.

(iv) D encrypts the bits of R with Mix_i's GM public key and multiplies them individually with bits of $\overset{e}{\mathbf{v}}_i$ to obtain $\overset{e}{\mathbf{p}}_i = \langle p_{i,1}, \ldots, p_{i,b}\rangle$ for $1 \leq i \leq 3$.

(v) Finally, D sends the four-tuple of ciphertext $\langle \overset{e}{\mathbf{p}}_1, R'_1, R_2, R_3 \rangle$ to Mix$_1$, $\langle \overset{e}{\mathbf{p}}_2, R_1, R'_2, R_3 \rangle$ to Mix$_2$ and $\langle \overset{e}{\mathbf{p}}_3, R_1, R_2, R'_3 \rangle$ to Mix$_3$.

Note that $\overset{e}{\mathbf{p}}_i$ is the GM encryption of the DC response M xor'ed with R, as GM is a homomorphic encryption scheme. Also, R is computed such that $R = R_1 \oplus R'_1 = R_2 \oplus R'_2 = R_3 \oplus R'_3$. Crucially, each Mix$_i$ receives an encrypted copy of $M \oplus R$, but does not have enough information to unmask (decrypt) M.

In summary, during this phase, each DC xor-encrypts its oblivious counters with a random value, and transmits that ciphertext plus shares of the xor'd random value to each of the three mixes.

Mix Noise Addition and Forwarding. Each mix on receiving the four-tuple ciphertext $CT = \langle C_1, C_2, C_3, C_4 \rangle$ from a DC (see step (v) above), checks the legitimacy of C_1. A legitimate GM encrypted value must have its Jacobi symbol equal to '+1', so a mix can easily and efficiently detect malformed responses. If the DC's response is not legitimate, a mix discards it. Otherwise, it decrypts C_1 using its GM private key and obtains the DC response M masked with R. The three mixes then synchronize the list of DCs that have responded. The master mix removes DCs that are not in list L_c. Let the total number of common DCs that have responded be c. To preserve the privacy of the DCs, the mixes collaboratively add n noisy four-tuples, where n is derived using Eq. 3.2 (where, $\delta = 10^{-6}/u$).

In order to make a noisy tuple and the DC responses indistinguishable, mixes use an efficient xor encryption:

Mix 1:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_2$ using pairwise common seed x_2 and $\mathcal{R}_3$ using pairwise common seed x_3.

(ii) Computes $\mathcal{R}'_1 = P \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$.

Mix 2:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_1$ using pairwise common seed x_1 and $\mathcal{R}_3$ using pairwise common seed x_3.

(ii) Computes $\mathcal{R}'_2 = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_3$.

Mix 3:

(i) Chooses random b-bit binary strings P using seed p, Q using seed q, $\mathcal{R}_1$ using pairwise common seed x_1 and $\mathcal{R}_2$ using pairwise common seed x_2.

(ii) Computes $\mathcal{R}'_3 = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2$.

Now, Q is indistinguishable from decrypted C_1, as Q is of the form $M \oplus R$ for some random M and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. Note that, each mix exactly knows only two of the three random vectors $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$. Therefore, each mix does not know R and hence, the noise that is being added. In other words, mixes add noise, but do not know the values of the noise that they contribute.

Each mix repeats steps (i) and (ii) until all n noisy tuples are generated. Each mix Mix$_i$ then arranges the c DC four-tuples and the n noisy four-tuples, row-wise into four matrices $\langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$. Let $M_i = \langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$, $1 \leq i \leq 3$. The mixes then shuffle the columns of each matrix in M_i independently, using common shuffling seed s. This shuffling prevents a DC from being identified, and eliminates a potential covert channel. Finally, each Mix$_i$ forwards M_i to the analyst. The master mix in addition forwards the list L_c of DCs that had taken part.

Aggregate Calculation. Upon receiving $M_i = \langle M_{i,1}, M_{i,2}, M_{i,3}, M_{i,4} \rangle$, $1 \leq i \leq 3$, the analyst first checks whether the mixes have tampered any DC responses by verifying if:

$$M_{1,1} \stackrel{?}{=} M_{2,1} \stackrel{?}{=} M_{3,1} \tag{5.3}$$

$$M_{2,2} \stackrel{?}{=} M_{3,2} \tag{5.4}$$

$$M_{1,3} \stackrel{?}{=} M_{3,3} \tag{5.5}$$

$$M_{1,4} \stackrel{?}{=} M_{2,4} \tag{5.6}$$

$$M_{1,2} \oplus M_{2,2} \stackrel{?}{=} M_{2,3} \oplus M_{3,3} \stackrel{?}{=} M_{3,4} \oplus M_{1,4} \tag{5.7}$$

If any of the equalities in Eqs. 5.3 through 5.7 does not hold, the analyst rejects the response. In such a case, attribution can be performed if exactly one of the mixes is malicious – the mix that does not agree on the equalities with the other two mixes is malicious.

If the equalities hold, then the analyst computes:

$$\bar{M} = M_{1,1} \oplus M_{1,2} \oplus M_{2,2} \tag{5.8}$$

Finally for every bin j, $1 \leq j \leq b$, the analyst computes the noisy aggregate $\mathbf{a}_j$ as follows:

$$\mathbf{a}_j = \sum_{k=1}^{c+n} \bar{M}[k, j] - n/2 \tag{5.9}$$

5.5 Privacy and Security Analysis

We next argue that HisTorϵ does not reveal any DC statistics to an adversary. We then discuss the privacy guarantees offered by HisTorϵ and evaluate the efficacy of various potential attacks.

5.5.1 Security Arguments

We argue the security of the protocol in parts: at the DC, at the mixes and at the analyst. The communication between any two participants (the DCs and the mixes, or the mixes and the analyst) are through TLS channels and are assumed to be secure against eavesdropping. Therefore, we consider the security of the statistics while they are stored on the participants and not while they are in transit on the network.

Claim 1. *Oblivious counters guarantee both confidentiality and integrity of the statistics being collected.*

Security Argument. An oblivious counter $\overset{e}{\mathbf{v}}$ is encrypted with a mix's GM public key. By the security of GM encryption, the oblivious counter cannot be decrypted without knowledge of the mix's private key. Moreover, a legitimate GM-encrypted value must have its Jacobi symbol equal to '+1', and hence a counter cannot be malformed by flipping random bits. In other words, GM encryption ensures that the values will only be decrypted as either a `0` or a `1`.

Therefore, the only way the counters can be tampered is by encrypting them to random legitimate GM-encrypted values. This scenario is equivalent to a malicious DC reporting erroneous data. Even in this case, each malicious relay can contribute at most 1 to each bin in its counter. The maximum influence over the aggregate is therefore bounded by the number of malicious DCs. Thus, it follows that the oblivious counters guarantee both confidentiality and integrity of the statistics being collected.

Claim 2. *A mix cannot learn DC statistics or manipulate them without detection.*

Security Argument. Each Mix$_i$ knows $\langle M \oplus R \rangle$ from the DCs. The $M \oplus R$ is a one-time pad with secret key R, where R is a random vector that can be obtained from any of the three pairs of random shares: $R_1 \oplus R'_1$, $R_2 \oplus R'_2$ or $R_3 \oplus R'_3$. Here, each Mix$_i$

has exactly one random share from each pair – R'_1, R_2, R_3 in case of Mix_1, R_1, R'_2, R_3 in case of Mix_2 and R_1, R_2, R'_3 in case of Mix_3. Therefore, each mix does not have enough information to unmask M and cannot learn any of the DCs' statistics.

A malicious Mix_i, $1 \leq i \leq 3$ can tamper with any of the four-matrices $M_{i,1}, M_{i,2}, M_{i,3}$ or $M_{i,4}$ it receives. We use a tainting technique to prove that the mixes cannot modify any of the matrices without detection. We say that a matrix is tainted with an non-zero impurity if a mix modifies it. Without loss of generality, let us assume that Mix_1 is malicious. Let W, X, Y, and Z be the non-zero impurities used to taint $M_{1,1}, M_{1,2}, M_{1,3}$, and $M_{1,4}$, respectively. All matrices from Mix_2 and Mix_3 are tainted with zero, as only Mix_1 is assumed to be malicious. If suppose, Mix_1 can manipulate the matrices without detection, then all the equalities in Eq. 5.3 - 5.7 hold. Therefore, Eq. 5.3, 5.5, 5.6 and 5.7 are tainted with W, X, Y, and Z as follows:

$$W = 0 \tag{5.10}$$

$$Y = 0 \tag{5.11}$$

$$Z = 0 \tag{5.12}$$

$$X \oplus 0 = 0 = 0 \oplus Z \tag{5.13}$$

From Equations 5.10 through 5.13, we can reach the conclusion that $W = X = Y = Z = 0$. This is a direct contradiction to our assumption that W, X, Y, and Z are non-zero impurities. Therefore, a mix cannot manipulate the DC responses without detection.

Claim 3. *An analyst cannot learn any DC statistics.*

Security Argument. The analyst can compute $\bar{M}$ from the mixes. $\bar{M}$ contains the DC responses and differentially private noise, and is shuffled column-wise using a cryptographic random seed s that is not known to the analyst. The differentially private noise is indistinguishable from a DC response by the way it is generated: A noise vector Q is of the form $M \oplus R$ for some random M, and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. Also the random vectors $\mathcal{R}'_1, \mathcal{R}'_2$ and $\mathcal{R}'_3$ are generated such that $\mathcal{R}_1 \oplus \mathcal{R}'_1 = \mathcal{R}_2 \oplus \mathcal{R}'_2 = \mathcal{R}_3 \oplus \mathcal{R}'_3 = R$. Therefore, the analyst sees a pseudorandom permutation of the DC responses and differentially private noise in the columns of $\bar{M}$. By the security of pseudorandom permutation, the columns of $\bar{M}$ cannot be distinguished from a random permutation with practical effort. Therefore, the analyst cannot learn any DC response from $\bar{M}$.

Claim 4. *A mix cannot learn the actual aggregate by subtracting the noise from the published aggregate.*

Security Argument. A mix adds a noise vector Q of the form $M \oplus R$ for some random M, and $R = P \oplus \mathcal{R}_1 \oplus \mathcal{R}_2 \oplus \mathcal{R}_3$. A mix exactly knows only two of the three random vectors $\mathcal{R}_1, \mathcal{R}_2$, and $\mathcal{R}_3$. So, a mix does not know R and hence the noise that is being added. Therefore, a mix cannot learn the actual aggregate by subtracting the noise from the published aggregate.

5.5.2 Privacy Analysis

Data Collector. The DCs can be malicious and report "junk" statistics. The maximum absolute distortion in the final aggregate result is bounded by the number of malicious DCs. This bound is much more lenient than PrivEx: the adversary needs to compromise many DCs to substantially distort the result provided by

HisTorϵ; in PrivEx, a single malicious data collector can significantly influence the aggregate result (see §5.1).

When two or more DCs collude, they learn no more information than would otherwise be available by pooling their knowledge. HisTorϵ protects a single honest DC's statistics even when all other DCs are malicious. The statistics collected at the honest DCs are preserved from the actions of a misbehaving DCs as long as the security of the GM encryption scheme remains intact.

Even when a DC colludes with a mix or an analyst, it learns no more information than what is already known to it.

Mixes. The mixes can be malicious and can report "junk" statistics or refuse to add noise. However, as long as at least two of the three mixes are honest, we can identify the malicious mix. Moreover, the malicious mix cannot learn any DC statistics as shown in §5.5.1.

Even when two non-colluding mixes are malicious, we can still detect such an attack, but cannot attribute it. Also, in either of these cases, the mixes do not know the noise added and cannot learn the actual aggregate.

However, when two or more malicious mixes collude, they can trivially learn all DC responses without detection. We discuss this threat in more detail in §5.9.

Analyst. The analyst cannot learn any DC's response as shown in §5.5.1. However, a malicious mix can share the shuffling seed s with the analyst. This allows the analyst to learn all the DC responses.

5.5.3 Attack Resilience

To complete our security analysis, we consider three types of attacks against HisTorϵ: compulsion attacks, denial-of-service (DoS) attacks, and correlation attacks.

Compulsion Attacks. A DC can be compelled to reveal its counters through a legal order or extra-legal compulsion. HisTorϵ mitigates this threat by encrypting the counters with the mixes' public key. A DC cannot decrypt its own oblivious counters.

The adversary can also compel the mixes to decrypt the DCs' statistics. However, each mix receives the client response masked with the random vector R, and does not have enough information to unmask it (each mix has either R_i or R'_i for $1 \leq i \leq 3$, but not both R_i and R'_i).

To successfully conduct a compulsion attack, the adversary would have to first compel the DC to release its oblivious counters, and then further compel a mix to perform the decryption. While such an attack is technically feasible, we imagine that compelling a mix to perform a decryption would garner significant attention; as a rough equivalent, this is analogous to compelling a Tor directory authority to release its signing keys.

The adversary may compel the analyst to release statistics before the analyst aggregates the result. However, this attack is fruitless since the analyst has only the noised data, with differentially private guarantees.

DoS Attacks. A DC can refuse to participate in a query or submit a malformed vector. This is easily mitigated by discarding the particular DC's response. Consequently, malicious DCs cannot cause denial-of-service in a round of HisTorϵ.

Mixes may also DoS HisTorϵ queries. The availability of the aggregate is guaranteed as long as at least two of the three mixes participate. Mixes are dedicated machines and hence it's highly unlikely that two mixes go down at the same time.

Correlation Attacks. An attacker can combine the collected statistics with some auxiliary information such as observations of a target user's network traffic

and perform a correlation attack. However, the differential privacy mechanism [42] provides strong privacy guarantees against such a threat.

5.6 A Practical Consideration: Guided Binning

When posing a histogram query, an analyst needs to define the bin widths. Determining useful bin widths for a histogram is highly subjective, and can be difficult if the analyst does not have an intuition as to the overall shape of the histogram.

Here, as a more practical contribution, we present a simple algorithm for partially automating this process. Conceptually, the algorithm operates by modifying the definition of bin widths (that is, by splitting and joining bins) in a sequence of epochs until a useful histogram is obtained. Since each epoch lasts one hour, our goal is to quickly converge on a useful bin width definition.

The analyst runs the guided binning algorithm until it gets a "satisfactory" histogram of the noised results. In our experiments, we find that we achieve a useful histogram after three or four epochs. Importantly, once a useful definition of bin widths is achieved for a given query – *e.g.*, the number of connections seen by guard relays – this definition tends to hold over time. That is, the guided binning algorithm is most useful when issuing a new type of query or when the results of a query indicate unexpected results.

We refer to each run of the guided binning algorithm as an iteration. The first iteration takes two parameters as input: b, the number of bins; and e, the total estimate of the statistics (e.g., total bandwidth, total number of client connections, *etc.*) being collected. Equal width bins are assigned for this first iteration – the lower bound of first bin is set to 0, a bin-width of $\lfloor e/b \rfloor$ is used for the first $b-1$ bins and the upper bound of last bin is set to e.

```
1  // w - New Bin Width List
2  // cur_w - Current Bin Width
3  // max - UB of Last Bin in Previous Iteration
4  proc gcdBinWidth(w, cur_w, max)
5  if !w then
6  |  g ← cur_w // List Empty
7  else
8  |  minw ← min(w)
9  |  g ← gcd(minw, cur_w)
10 |  if g = 1 or g < min(minw, cur_w) then
11 |  |  i ← 1
12 |  |  sqrt ← √minw + 1
13 |  |  // compute greatest factor of minw lesser than or equal to cur_w
14 |  |  while i ≤ sqrt do
15 |  |  |  if minw % i = 0 and minw / i ≤ cur_w then
16 |  |  |  |  g ← minw/i
17 |  |  |  |  break
18 |  |  |  end
19 |  |  |  i ← i + 1
20 |  |  end
21 |  |  // g is 1 or auxiliary bins > 15000
22 |  |  if i = sqrt + 1 or max/g > 15000 then
23 |  |  |  g ← minw
24 |  |  end
25 |  end
26 end
27 return g
```

Procedure GCDBinwidth

```
1  // g - GCD returned by gcdBinwidth()
2  // cur_w - Current Bin Width
3  proc adjustBinWidth(g, cur_w)
4  if cur_w < g then
5  |   cur_w ← g
6  else
7  |   // cur_w not a multiple of g
8  |   if cur_w % g ≠ 0 then
9  |   |   // make cur_w a multiple of g
10 |   |   if (cur_w % g) < (g − (cur_w % g)) then
11 |   |   |   cur_w ← cur_w − (cur_w % g)
12 |   |   else
13 |   |   |   cur_w ← cur_w + (g − (cur_w % g))
14 |   |   end
15 |   end
16 end
17 return cur_w
```

Procedure AdjustBinwidth

If more iterations are needed, the next iteration uses the bin width distribution and the noised result of the previous iteration to obtain a more optimal bin width distribution. The guided binning algorithm first computes the mean value, k of the noised distribution. Then, in the lexical ordering of bins, starting from bin 1, it splits all bins that have a value r_i greater than k into $\lfloor r_i/k \rfloor$ bins. The algorithm also merges all consecutive bins that have a value less than k until their combined value does not exceed k.

When a bin is split or when bins are merged, the algorithm executes the GCD-Binwidth procedure. The GCDBinwidth procedure chooses an optimal GCD value g (of the bin widths up to that point) such that the total number of auxiliary bins (see §5.3.2) is less than 15000. Then the guided binning algorithm executes the AdjustBinwidth procedure that makes the current new bin width a multiple of the optimal GCD g.

The algorithm can be terminated during any iteration in which the analyst obtains a "satisfactory" histogram of the noised results.

5.7 Implementation and Evaluation

We constructed an implementation of HisTorϵ in Python to verify our protocols' correctness, assess the utility of the noised aggregates, and measure the system's overheads. We implement GM encryption using the Python libnum library [5] with a modulus size of 1024 bits. Our PRF is based on AES in CFB mode, supported by the PyCrypto cryptographic library. Mixes perform shuffle operations by applying the Fisher-Yates algorithm, using the AES-based PRF as its source of randomness.

As a system-wide parameter, we set $\epsilon = 1$ and $\delta = 10^{-12}$ for all our experiments unless otherwise indicated. We note that this offers more privacy than the experimental setting $\epsilon = 5$ used by Chen et al. [42]. For histogram queries, we semi-automate the process of selecting appropriate bin widths using the human-guided bin splitting algorithm described in §5.6.

Experiments were carried out on a 16-core AMD Opteron machine with 32GB of RAM running Linux kernel 3.10.0. Our implementation of HisTorϵ is currently single-threaded. Although HisTorϵ's computational costs are insignificant (see §5.7.4), certain operations are embarrassingly parallel – in particular, encrypting and decrypting the elements of the $\vec{\mathbf{v}}^{e}$ vectors – and could likely further benefit from parallelization.

We instantiated three HisTorϵ mix instances, one HisTorϵ analyst instance, and all DCs on our 16-core server. In a real deployment, these instances would all be distributed. Google Protocol Buffers [138] were used for serializing messages, which were communicated over TLS connections between HisTorϵ components.

We use Python's default TLS socket wrapper for securing communication. All communication took place over the local loopback mechanism (localhost).

Our experiments do not run actual Tor relay or client code. As explained in more detail next, we derive our unnoised statistics from existing published data sets from the Tor Compass and Tor Metrics Portal. This data is used as input to our DC instances, which then run HisTorϵ protocol to enable the analyst to obtain (noised) aggregate results.

5.7.1 Queries and Data Sets

We evaluate HisTorϵ by considering three histogram queries: the number of client connections as observed by Tor guards, the amount of bandwidth used by Tor guards, and the amount of exit bandwidth used by Tor exit relays. As our ground truth, we use data from both the Tor Compass and the Tor Metrics Portal.

Number of Client Connections. For the number of client connections, each Tor guard acts as a DC. In total, we instantiate 1839 DCs – the total number of guards reported by the Tor Compass with a non-zero selection probability.

We derive our "ground truth" by considering the total number of direct users (T) connecting to Tor as reported by the Tor Metrics Portal over the period of July 26th through July 30th, 2016. We assign $p_i \cdot T$ client connections to each guard relay i, where p_i is the guard selection probability for guard i as reported by the Tor Compass.

Bandwidth Used by Guards/Exits. Similarly, as our ground truth for the bandwidth observed by guards (resp. exits), we consider the total guard (resp. exit) bandwidth (B) reported by the Tor Metrics Portal over the same five-day time period. Each guard (resp. exit) acts as a DC, and is assigned a bandwidth cost of $(p_i \cdot B)$,

where p_i is the selection probability of the guard (resp. exit). We instantiate 1839 DCs when measuring guard bandwidth, and 924 DCs in the case of exit bandwidth. The latter is the number of exits reported by the Tor Compass that have a non-zero selection probability.

We do not argue that the above procedures yield perfect ground truth. We apply them to derive a gross approximation of the distributions of client connections and bandwidths which can then be used to test the efficacy of HisTorϵ under near-real-world conditions. When deployed, HisTorϵ allows for much more accurate and fine- grained statistics reporting than offered by the Tor Metrics Portal.

5.7.2 Accuracy

Figure 5.5 shows the returned histograms for the three queries when applied to the Tor datasets. The Figure plots the results of the histogram query after three iterations of the guided binning algorithm (see §5.6). Other iterations exhibited similar accuracy (as measured by the difference between the noised and unnoised distributions), but had arguably less useful bin definitions.

HisTorϵ reports the "Noised" values shown in Figure 5.5. As is clear from the Figure, these noised values closely resemble those of the unnoised ("Actual") ground truth data. Looking at just the "Noised" values, an analyst can clearly obtain useful information about the distributions of client connections, guard bandwidths, and exit bandwidths.

As a more quantifiable indicator of the *closeness* between the noised and unnoised distributions, we consider both the coefficient of determination (also called the R^2 distance) and the Bhattacharyya Distance [29]. The latter measures the divergence between two probability distributions and ranges from 0 (identical distri-

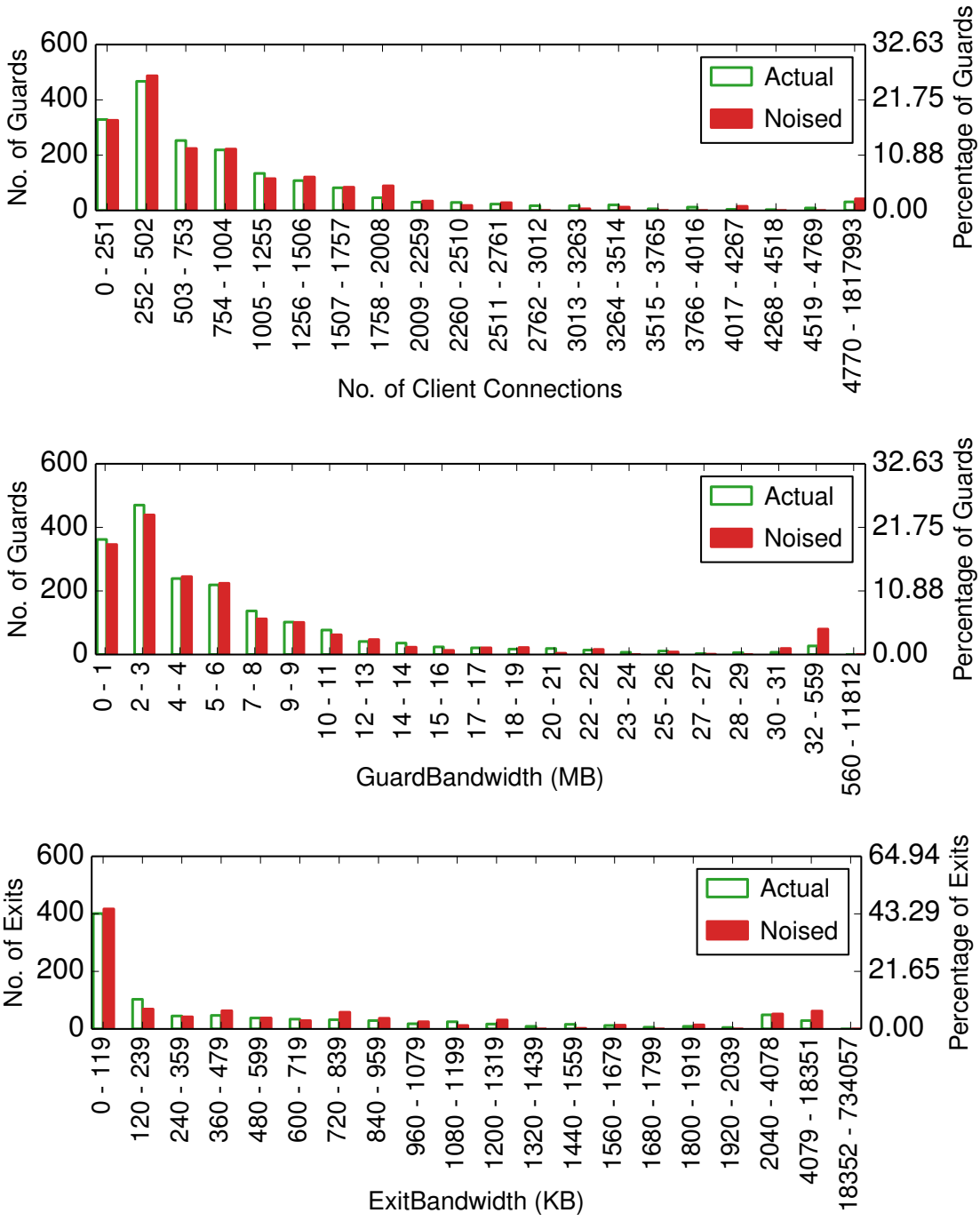

Figure 5.5: Aggregate histogram results returned by HisTorϵ. The aggregate results returned when the analyst issues a query for the number of client connections as observed by guards (*top*), the amount of bandwidth used by guards (*middle*), and the amount of bandwidth used by exit relays (*bottom*). Also shown is the unnoised distribution ("Actual").

Table 5.1: Distance between "actual" distribution and "noised" distribution.

Distance function	No. Client Conn	GuardBW	ExitBW
R^2	0.98466	0.98290	0.96970
Bhattacharyya	0.01820	0.01179	0.02542

Distances between the "actual" and "noise" distributions in Figure 5.5.

butions) to ∞. An R^2 value of 1 indicates perfect prediction. When no correlation exists, $R^2 = 0$. Table 5.1 reports the distances between the actual and noised distributions. Our results highlight that even with a conservative setting of $\epsilon = 1$, HisTorϵ produces highly accurate aggregates.

The tradeoff between accuracy and privacy is governed by the choice of ϵ. We explore this space by varying ϵ between 0.2 and 2.0 for the connection count query. As shown in Figure 5.6, we find that varying ϵ has little effect on accuracy. The overall variation in R^2 (resp. Bhattacharyya) distance between $\epsilon = 0.2$ and $\epsilon = 2.0$ was only 0.315 (resp. 0.098).

5.7.3 Bandwidth Overhead

HisTorϵ incurs communication overhead by transmitting encrypted counters between the DCs and the mixes, and encrypted matrices between the mixes and the analyst. To be practical, a statistics gathering system should impose a low communication overhead for the DCs, since relays are already a bandwidth-limited resource in Tor [52]. We envision that mixes and the analyst are dedicated resources for HisTorϵ, and our goal is to not incur unreasonable bandwidth requirements for these components.

We explore HisTorϵ's bandwidth costs by varying the number of bins b in a query. The values of the bins for the type of query (class vs. histogram) do not affect

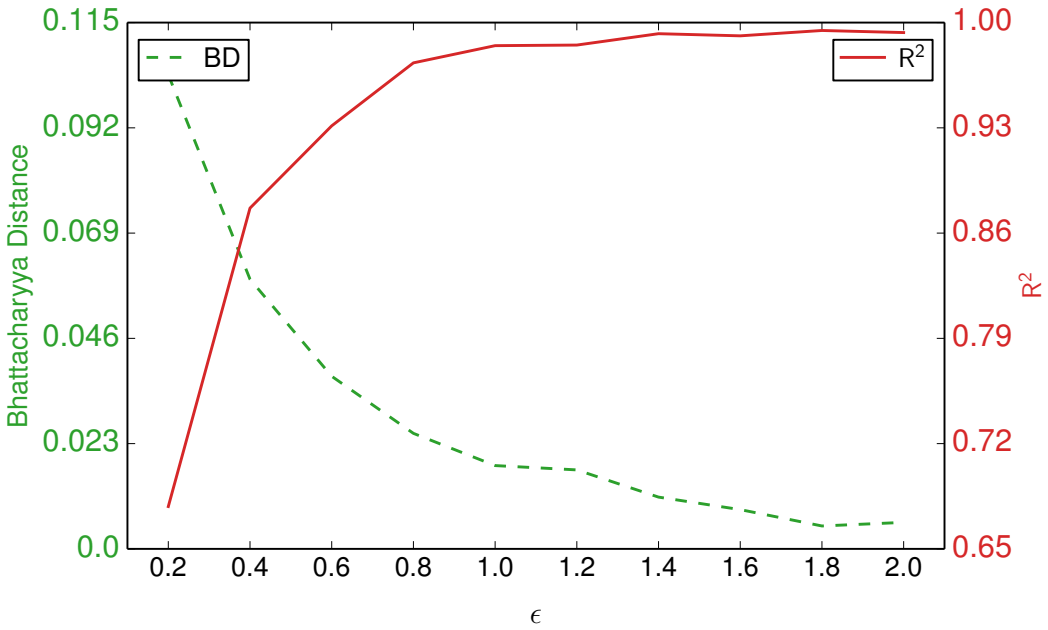

Figure 5.6: Distance between "actual" and "noised" distribution. The distance between the actual and noised distributions, as measured by the Bhattacharyya distance (left y-axis) and R^2 (right y-axis).

the communication cost, as the DCs only transmit $\overset{e}{\mathbf{v}}$ (and not $\overset{e}{\boldsymbol{\nu}}$) for both query types. In our bandwidth measurements, we fix the number of DC relays at 1839.

Figure 5.7 shows the average communication costs for a DC, mix, and analyst. For up to 80 bins, the communication cost for each DC is fairly modest and is approximately 150 KB per hour (or about 42 Bps). Generally, we anticipate the number of bins to be around 20, although this can vary depending upon the analyst's query. As a potential point of interest, the histograms shown in Figure 5.5 were derived using the guided binning process, and resulted in 20, 21, and 20 bins (from left to right).

Even when the number of bins is quite large (1280), a DC's communication cost is only 2.4 MB over the course of the hourlong epoch – or 0.67 KBps.

The communication costs are greater for the mixes and the analyst. With 40 bins, each mix consumes 47.8 MB of bandwidth per hour, while the analyst uses

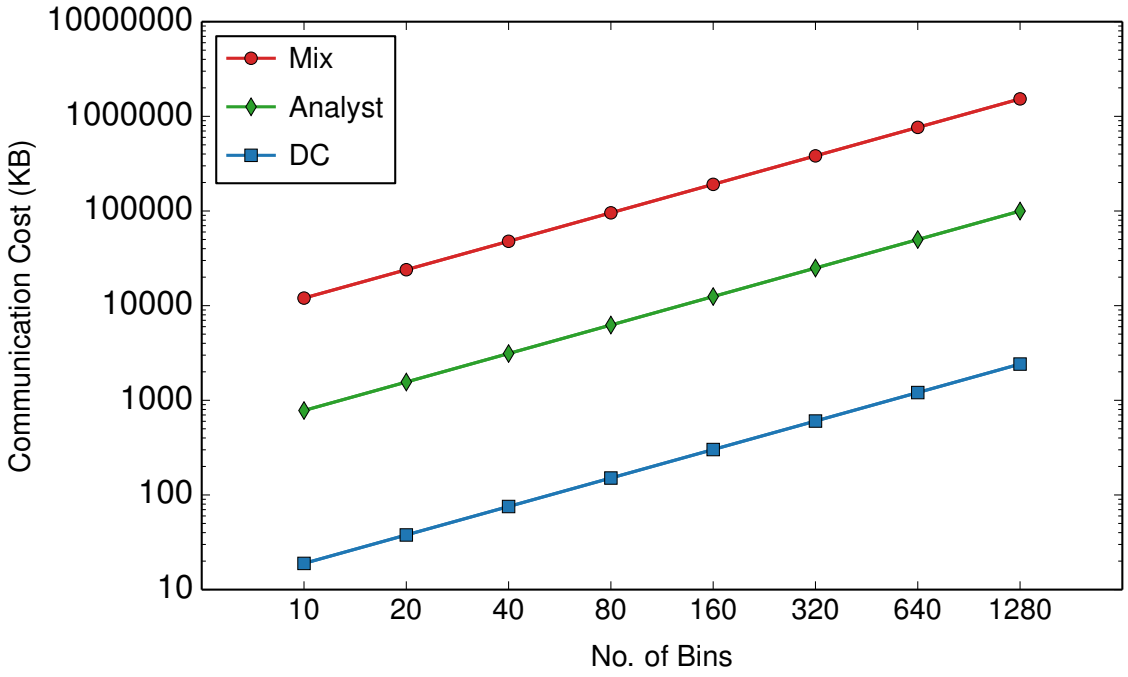

Figure 5.7: HisTorϵ communication cost. The communication cost per epoch (y-axis) as a function of the number of bins (b; shown on the x-axis). Both axes are plotted in log scale.

3.1 MB. In our largest binning scenario with 1280 bins, each mix consumes 1.5 GB of bandwidth each hour, or 424.7 KBps; the analyst requires 27.7 KBps.

In summary, we find that for modest number of bins (20-40), HisTorϵ incurs very little bandwidth overhead, and can thus support multiple concurrent queries.

5.7.4 Computation Overheads

To understand the computation costs of HisTorϵ, we perform a series of microbenchmarks. In our measurements, we consider the connection counting histogram query with twenty bins and 1839 DCs.

The left side of Figure 5.8 shows the distribution of the processing overheads for the DCs. The operations involved in maintaining the oblivious counters are initialization, incrementing, and mapping (from $\overset{e}{\boldsymbol{\nu}}$ to $\overset{e}{\mathbf{v}}$). The total costs of these

operations is, in the worst case, less than two seconds per hour (on a single core). Additionally, the DCs generate the random vectors $\mathbf{R_i}$ and $\mathbf{R'_i}$ (such that $\mathbf{R} = \mathbf{R_i} \oplus \mathbf{R'_i}$), incurring a worst case processing overhead of approximately 7.2 ms per hour.

The performance overheads for a mix are shown in the center of Figure 5.8. To explore the range in overheads, we repeat our query 100 times and plot the range of processing costs incurred by the mix over all runs. Here, the operations consist of GM decryption, the generation of noise records, and shuffling the matrices. Each hour, in the worst case, a mix spends approximately one minute of computation for a single query. Using a single core, a mix could thus support at worst 60 simultaneous queries per hour.

The analyst's processing times are shown in the right side of Figure 5.8. As with the mix, we show the results over 100 iterations of the query. The most burdensome operation is verification of the three matrices. This consumes less than three seconds of processing, in the worst case, per hour.

Overall, the processing costs of operating HisTorϵ is negligible for the relays (DCs) and analyst, and manageable for the mixes.

5.8 Related Cryptographic Protocols

In this section, we briefly review some of the cryptographic protocols which are related to the design of HisTorϵ.

HisTorϵ uses a slight modification of the (ϵ, δ)-differential privacy scheme of Chen et al. [42]. Chen et al. use a single mix, which they call a *proxy*. This allows a malicious proxy to undetectably alter the data and cause the analyst to reach an inaccurate aggregate result. We detect such manipulation in HisTorϵ by adding redundancy across three mixes. HisTorϵ detects such tampering if at least one of

the mixes is honest, and can attribute the misbehavior if two of the three mixes are honest. Additionally, the scheme of Chen et al. is vulnerable to "compulsion attacks", wherein a relay operator can be forced to reveal its counter through a subpoena. HisTorϵ uses oblivious counters to mitigate such risks.

HisTorϵ is also partially inspired by SplitX, which executes differentially private queries over distributed data [43]. Like SplitX, HisTorϵ uses the (ϵ, δ)-differential privacy scheme of Chen et al. and uses xor-based encryption to distribute secret shares to separate mixes. In SplitX, both the mixes and the aggregator are assumed to be honest-but-curious. In HisTorϵ, we are able to tolerate a malicious mix by redundantly encoding information in secret shares.

5.9 Discussion and Limitations

In this section, we discuss practical aspects of deploying HisTorϵ, as well as some of the system's limitations.

Detectability, Attribution and Suitability of the Threat Model. A malicious mix may attempt to manipulate the results of a query by modifying the inputs it receives from the DCs. As we discuss in §5.4 and §5.5, an analyst can *detect* that misbehavior occurred if at least one of the mixes is honest. Since data is replicated across all three mixes, we can additionally *attribute* the misbehavior to a specific malicious mix if exactly two of the three mixes are honest – the malicious mix will be revealed through its non-conforming output.

The difficulty with performing attribution is that the cases of one malicious mix vs. two malicious mixes can be indistinguishable if, in the latter case, the two mixes perform identical manipulations. Unless it is readily apparent through some

other mechanism which mix(es) has been compromised, a reasonable solution once misbehavior is detected is to re-evaluate the security of all three mixes.

More generally, mixes should be carefully selected, since collusion between two or more dishonest mixes compromises data privacy. This is, to some degree, similar to Tor's existing quasi-centralized notion of trust: if a majority of the Tor directory authorities are compromised, then Tor offers no anonymity protections since the directories could advertise only the existence of malicious relays.

Like directory authorities, mixes must therefore be chosen carefully. We envision that the maintainers of the Tor Project could selectively grant permission to operate HisTorϵ mixes to parties. Or, to keep the existing level of trust, the directory authorities could additionally operate mixes.

From the perspective of integrity, it may at first blush seem that HisTorϵ and PrivEx offer similar guarantees – that is, certain nodes must behave honestly to ensure integrity of the query results. We argue that the integrity guarantees offered by HisTorϵ are significantly stronger, since in PrivEx, a single relay can significantly perturb the results of a query. Successfully compromising PrivEx statistics gathering is thus a fairly simple operation since there are no barriers to operating a relay. In contrast, HisTorϵ removes the necessity to trust the data collectors, and instead relies on a much smaller set of nodes (i.e., mixes). Additionally, so long as a single mix is honest, query tampering can be trivially detected.

Selection of ϵ. A consistent problem in schemes that apply differential privacy is selecting an appropriate value of ϵ. In our experimentation, we apply the same conservative value as existing work [106] and set $\epsilon = 1$.

Since ϵ is a relative and not an absolute measure of privacy, an interesting area of future work is to derive *per-query* values of ϵ that are guaranteed to protect individuals with some fixed probability. [95] provide one such construction for ϵ-differential

privacy. Incorporating this selection process into HisTorϵ (which provides (ϵ, δ)-differential privacy) is an exciting potential future research direction.

In the current implementation of HisTorϵ, the analyst communicates its choice of ϵ to the mixes. Since the number of noise records is proportional to ϵ^{-2}, large values of ϵ offer little security while too small values of ϵ have little benefit to privacy while incurring potentially enormous communication costs. To provide a simple sanity-check, a real-world deployment of HisTorϵ could eustablish system-wide parameters ϵ_{max} and ϵ_{min} that bound the analyst's choice.

Privacy Budget. (ϵ, δ)-differential privacy schemes impose a *privacy budget* whose balance is decremented as a function of δ and ϵ for each issued query. This budget is defined in terms of a static database $\mathcal{D}$ over which queries are issued. In HisTorϵ, counters are zeroed after each epoch, effectively resulting in a new database. This thus significantly reduces the risk of exceeding the privacy budget.

Unfortunately, for certain query types, there may be dependencies in the statistic of interest between epochs that violates this assumption of independence. This further motivates a careful selection of ϵ to minimize this privacy risk.

5.10 Summary

In this chapter, we present HisTorϵ, a distributed statistics collection system for Tor. HisTorϵ provides strong integrity guarantees, and ensures that a small colluding group of malicious data collectors has negligible impact on the results of statistics queries.

In addition to ensuring integrity, HisTorϵ provides strong privacy guarantees as long as malicious mixes do not collude with a dishonest analyst. HisTorϵ also achieves resistance to compulsion attacks through use of novel oblivious counters.

We demonstrate using real-world data sets and realistic query workloads that HisTorϵ enables highly accurate statistics aggregation, with small bandwidth and computational overheads.

In the next chapter, we introduce Private Set-union Cardinality (PSC), a differentially private statistics gathering system that efficiently aggregates the count of unique items recorded across a set of data collectors privately.

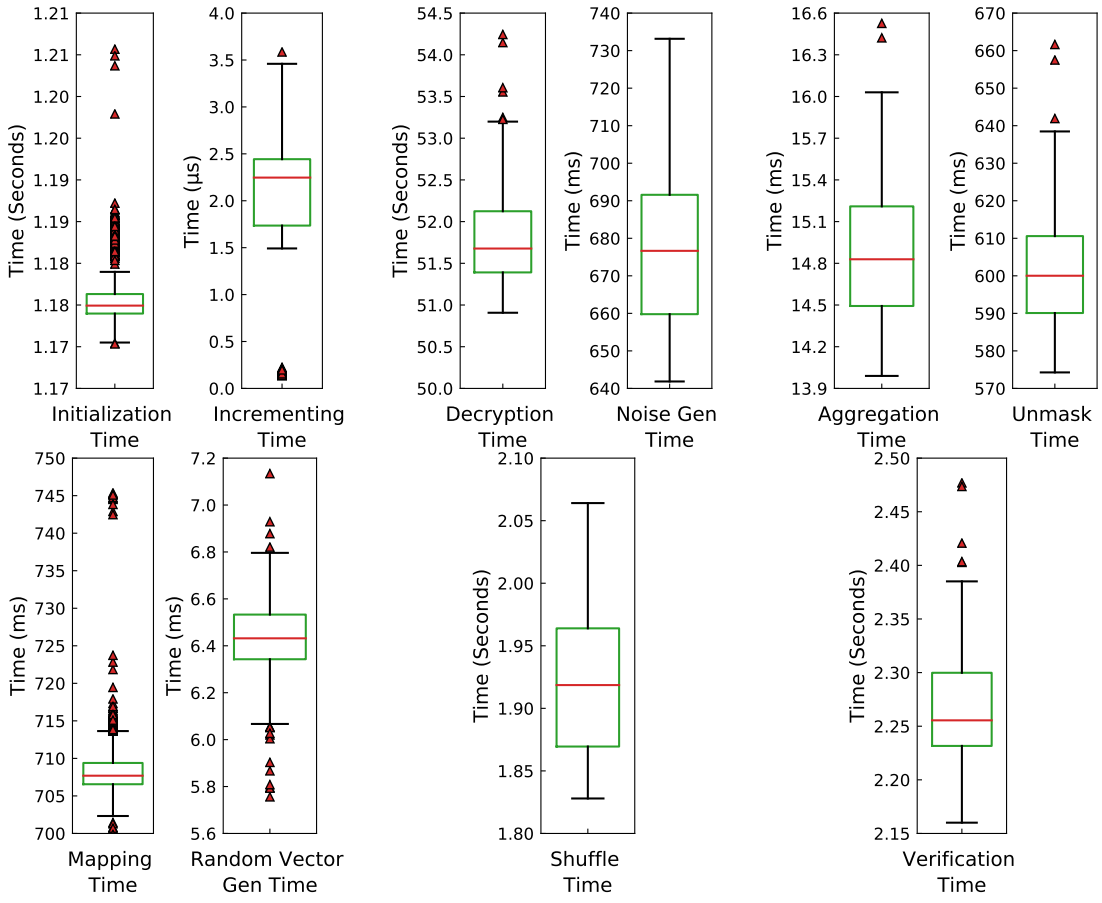

Figure 5.8: HisTorϵ computation cost per operation. Microbenchmark results for the DCs (left), mixes (center), and analyst (right). Boxes denote the lower and upper quartile values, with a line at the median. The "whiskers" show the range of the data within 1.5xIQR from the lower and upper quartiles. Outliers are indicated with triangles.

Chapter 6

Distributed Private Set-Union Cardinality

In this chapter, we describe Private Set-union Cardinality (PSC), a differentially private statistics gathering system for privacy-preserving distributed systems, such as anonymity networks. Unlike HisTorϵ, and prior secure measurement systems [62, 80], PSC efficiently aggregates the count of unique items recorded across a set of Data Parties (DPs) privately, without exposing any information other than the count. For example, while PrivEx, PrivCount, and HisTorϵ answer the question – *how many clients were observed entering the Tor network?*, PSC answers the (perhaps) more useful question of *how many unique clients were observed on Tor?*

An important property of PSC– and one that is shared with PrivEx, PrivCount, and HisTorϵ– is that the Data Parties collect statistics *obliviously* using encrypted counters, that they cannot read on their own. This makes PSC resistant to the so called "compulsion attack" in which the DPs can be compelled to release sensitive information under pressure (*e.g.*, in the form of a subpoena).

PSC is provably secure in the Universal Composable (UC) framework [35] against an active adversary that controls all but one of the aggregation parties. It also provides security against adaptive corruption of the Data Parties, which prevents them from being victims of targeted compromise. Moreover, PSC provides

full accountability against malicious parties, which in extension makes it Denial of Service (DoS) resistant (since malicious parties cannot cause the protocol to abort, without detection). Furthermore, to ensure safe measurements, the output satisfies differential privacy guarantees.

In the following sections, we describe the PSC protocol (in two parts). First, we begin with a simpler version of the PSC protocol, called the *base* PSC protocol [67]. However, the base protocol does not provide full accountability against malicious parties. We next describe the complete version of the PSC protocol with full accountability and present a proof-of-concept implementation of it, with *relaxed* security guarantees. Finally, we demonstrate that, PSC operates with low computational overhead and reasonable bandwidth. In particular, for reasonable deployment sizes, the protocol runs at timescales smaller than the typical measurement period and is well-suited for private measurements in a distributed setting (*e.g.*, anonymity networks).

6.1 Private Set-Union Cardinality Problem

Recall from §1.3 that *private set-union cardinality* estimation is the problem of counting the number of unique private values across distributed data owners. More formally, private set-union cardinality answers how many unique items are recorded across a set of Data Parties while preserving the privacy of the inputs. Specifically, if there are d DPs and each DP_k contains a set of zero or more items $\mathcal{I}_k$, we want to know $|\cup_{k=1}^{d} \mathcal{I}_k|$ without exposing any element of any item set to an active adversary. In instances where exact counts may reveal sensitive information, private set-union cardinality may naturally be combined with differential privacy to provide noisy answers.

Motivation. Private set-union cardinality is useful in many settings. For example, in the context of anonymity networks, it can determine how many *unique* users participate in the service, how many unique users connect via a particular version of client software, and how many unique destinations are contacted. Maintainers of anonymity networks can also use private set-union cardinality to determine how users regularly connect to the network (*e.g.*, over mobile connections, through proxies, *etc.*) and how the network is used (*e.g.*, the length of time that users spend on the network in a single session). In the next chapter, we describe a number of measurements using PSC of the Tor anonymity network.

For structured overlay networks such as Chord [127], private set-union cardinality can determine the number of unique clients in the network and the number of unique lookups, without exposing users' identities.

For the web, private set-union cardinality can serve as a building block to determine the number of shared users among several sites, without revealing those users' identifiers.

More generally, private set-union cardinality allows researchers and administrators of distributed systems to better understand how such systems are being accessed and used. It enables system designers to make informed, data-driven decisions about their systems based on actual usage safely when privacy is required.

6.2 Background

Before describing PSC, we briefly review some background that is necessary for understanding our algorithm.

Discrete-Log Zero-Knowledge Proofs. PSC uses zero-knowledge proofs for demonstrating knowledge of and relationship among the discrete logarithms of certain

group elements. The elements are from some group G of order q, and the discrete logs are with respect to some generator $g \in G_q$ (*e.g.*, x is the discrete log of $y = g^x$, $x \in \mathbb{Z}_q$). In general, a zero-knowledge protocol [73] is a method by which a prover $\mathcal{P}$ can prove to a verifier $\mathcal{V}$, the truth of some statement without revealing any information apart from that truth. Here, the statement may be, for example, the existence or knowledge of a witness to membership in a language. Sigma protocols (*i.e.*, three-phase interactive protocols starting with a commitment by the prover, followed by a random challenge from the verifier, and ended by a response from the prover) exist for the discrete-log statements that PSC proves in zero-knowledge. Such sigma protocols can be made non-interactive via the the Fiat-Shamir heuristic [68] (secure in the random-oracle model), in which the random challenge is generated by the prover by applying a cryptographic hash to the commitment.

Verifiable Shuffles. PSC uses *verifiable shuffles* [111]. A re-encryption verifiable shuffle takes ciphertexts as inputs, outputs a permutation of a re-encryption of those ciphertexts, and provides a verifiable proof that the output is a valid permutation of a re-encryption of the input. A verifiable shuffle has two security requirements – privacy and verifiability. Privacy requires that an honest shuffle does not reveal any information about the secret permutation. Verifiability requires that any attempt by a malicious shuffle to tamper with the output must be detectable. Several verifiable shuffles have been proposed [27, 70, 78, 110]. As with the discrete-log proofs, the verifiable shuffes with interactive proofs can be made non-interactive using the Fiat-Shamir heuristic [68], which is secure in the random-oracle model.

Secure Broadcast. PSC uses a secure broadcast communication. The security requirement here, is broadcast with abort [74], a slightly weaker notion than Byzantine agreement. This guarantees that there exists some consensus output x such that each honest party that terminates, outputs x or aborts. Given a PKI, broadcast with

abort can be achieved with the two-round echo-broadcast protocol. In this protocol the sender sends a signed message to all the receivers, and every receiver appends its own signature to the received message and sends it to all other receivers. A receiver accepts the message, if it received the same message in all cases and with valid signatures, and otherwise it aborts.

6.3 Overview

PSC securely computes the cardinality of unique items distributed across a set of Data Parties (DPs). We first describe the *base* PSC protocol [67] (without accountability), which is UC-secure [35] and provides strong security and privacy guarantees. The base protocol has two phases:

In the *data collection phase*, the DPs record observations in a vector of encrypted counters. These observations correspond to statistic of interest – for example, the count of unique websites observed by the exit relays in an anonymity network (*e.g.*, Tor), or the count of unique clients observed by the network's entry relays. As discussed below, the counters are maintained obliviously, meaning that even the DPs, that maintain these counters cannot reveal the plaintext.

Once the data has been collected, the *aggregation phase* proceeds as a series of cryptographic operations that, *in toto*, produces the cardinality of the union of the DPs' observations.

For clarity, the operation of these two phases is summarized below. The full details are found in §6.4.

Participants and Threat Model. The participants in the system are the d DPs and m Computation Parties (CPs). The CPs are dedicated servers that perform series of

cryptographic operations to compute the private set-union cardinality of the DPs' observations.

Informally, the privacy requirements are (i) no information from an honest DP is ever exposed and (ii) the aggregated result (*i.e.*, the cardinality of the union) reveals no information about any individual data. Here, we do not attempt to prevent malicious parties from causing the protocol to abort.

Although, the base PSC protocol can be easily disrupted by malicious (or misbehaving parties), it still provides strong security and privacy guarantees (described in §6.4.7).

Data Collection Phase and Encrypted Counters. Each DP maintains its dataset as a vector of encrypted counters. We assume a mapping between possible values in the DPs' itemsets and some finite range of integers. This mapping may be trivially realized by hashing the itemsets' values. More concretely, let $\mathcal{H}$ be a hash function that maps itemset values to integers in the range $[0, b-1]$. Conceptually, each DP stores a b-bit encrypted bit vector, where an encrypted `1` indicates that the DP has the corresponding item in its itemset.

An important property of our system – and one that is shared with existing work on privacy-preserving measurement systems [62, 80, 101] – is that the counters be stored *obliviously*. That is, after an initialization step, the DPs discard the keys used to encrypt the counters. They do, however, have the ability to transform any element in their encrypted counter to an (encrypted) `1`, regardless of its previous (and unknowable) value. This construction allows the DPs to update the counters (e.g., to log the observation of a new client IP address) while maintaining resistance to *compulsion attacks*. That is, even under pressure to do so (*e.g.*, in the form of a subpoena), DPs cannot release the plaintext of their encrypted counters.

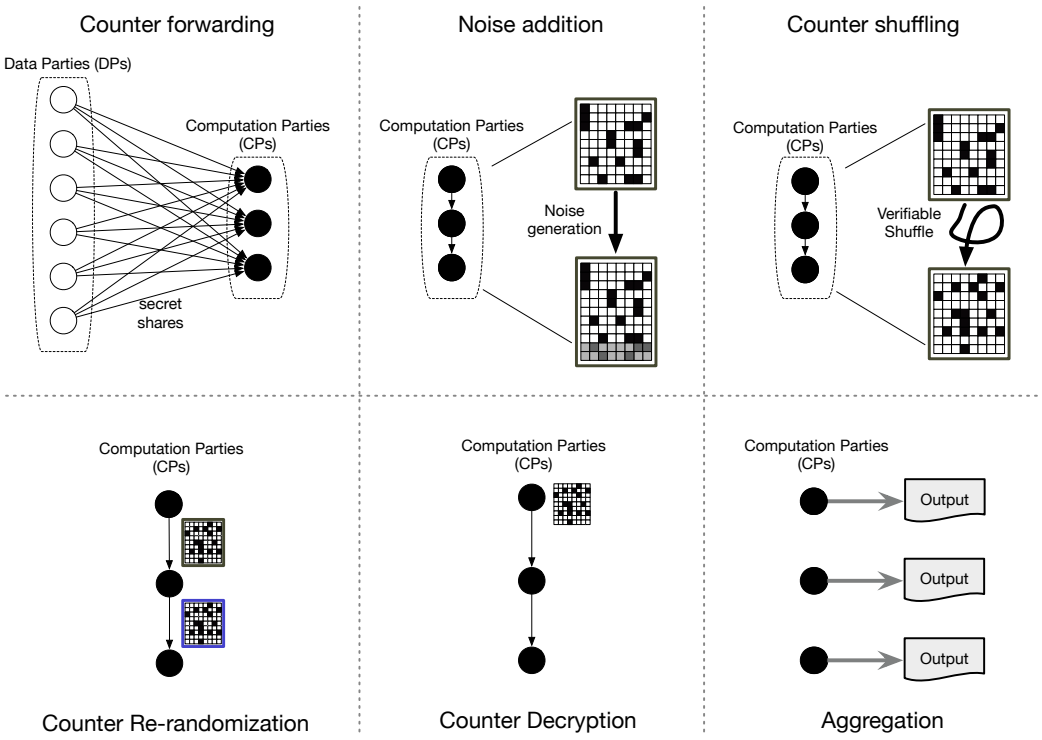

Figure 6.1: PSC protocol overview. A simplification of the major steps of the private set-union cardinality protocol. Random shares of the encrypted counters are first forwarded from the DPs to the CPs (a). The CPs then add noise (b) and verifiably shuffle the counters (c). The encrypted counters are re-randomized to prevent linkability (d), and then decrypted (e) and finally aggregated (f) to yield the final output.

Aggregation Phase. During the aggregation phase, the DPs forward their encrypted counters to the CPs, which in turn perform a series of steps to securely compute the cardinality of the union.

In more detail, the aggregation phase proceeds as follows:

1. *Counter forwarding:* Each DP creates secret shares of its encrypted counters and sends those shares to each of the CPs.

2. *Noise generation:* Upon receiving the shares, the CPs collaboratively compute encrypted counters by homomorphically adding encrypted shares. The CPs

then collaboratively generate n encrypted noise counters according to Eq. 3.2 and add the noise to their list of counters.

3. *Counter shuffling:* The CPs next perform a collaborative verifiable shuffle, which mixes the noise and non-noise counters and obscures the mapping between a position in the vector and a particular itemset value.

4. *Counter re-randomization:* To unlink the output counter values from the input values, each CP successively re-randomizes the values of the (shuffled) encrypted counters. This prevents a malicious DP from recognizing any output counter by "marking" it with a specific input value.

5. *Counter decryption:* After the counters have been re-randomized, each CP removes its layer of encryption, eventually exposing the plaintext (but shuffled) counters.

6. *Aggregation:* Finally, the CPs forward the shuffled counters to a single CP, which then computes the sum (*i.e.*, counts the 1s in the vectors) and reports the result.

This process is summarized in Figure 6.1 and described more fully next.

6.4 Base PSC Protocol

PSC's goal is to compute the cardinality of the union of private datasets distributed across a set of DPs with differential privacy guarantees. We introduce the base PSC protocol [67] by describing the preliminaries, the data collection and aggregation phases in detail, and few practical optimizations.

6.4.1 Preliminaries

PSC makes use of many cryptographic primitives. A primary primitive is ElGamal encryption [63], and PSC takes advantage of several properties of it, including distributed key generation and decryption, multiplicative homomorphism, re-encryption, and re-randomization of plaintext. Let G be a Diffie-Hellman group (*i.e.*, a finite cyclic group) for use by the ElGamal cryptosystem. Let q be the order of G, and let g be a generator of G. Let $x \xleftarrow{R} S$ denote an element x chosen uniformly at random from a set S. An ElGamal private-public keypair (x, y) is generated by choosing $x \xleftarrow{R} \mathbb{Z}_q$ and setting $y = g^x$. A message $m \in G$ is encrypted using the public key y, by choosing $r \xleftarrow{R} \mathbb{Z}_q$ and producing the ciphertext pair $(c_1, c_2) = (g^r, y^r m)$. A ciphertext pair (c_1, c_2) is decrypted using the private key x, as $m = c_2/c_1^x$. The ElGamal cryptosystem is multiplicatively homomorphic, *i.e.*, for any two messages $m_1, m_2 \in G$, $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$, where $E(\cdot)$ denotes encryption using the public key y. An ElGamal ciphertext (c_1, c_2) pair is re-encrypted (*i.e.*, generating a different ciphertext for the same plaintext) by choosing a randomization parameter $r \xleftarrow{R} \mathbb{Z}_q$ and producing $(g^r c_1, y^r c_2)$. A ciphertext is re-randomized by choosing a randomization parameter $r \xleftarrow{R} \mathbb{Z}_q^*$ and producing (c_1^r, c_2^r), which, if the original encrypted value is $z \neq g^0$, makes the value uniformly random in $G \backslash \{g^0\}$, and otherwise doesn't change it.

We assume PSC has access to some cryptographic primitives as "black-boxes" to provide ideal functionalities:

1. *Secure point-to-point communication functionality* ($\mathcal{F}_{SC}$)*:* Delivers messages from one party to another with confidentiality and authenticity.

2. *Secure broadcast-with-abort functionality* ($\mathcal{F}_{BC}$)*:* Guarantees that the same message is delivered to all honest parties that don't abort.

Such functionalities can be realized in the Universally Composable (UC) framework [35], and the details can be found in the appendix of the research work by Fenske et al. [67].

We also assume PSC has access to the following zero-knowledge proofs as "black-boxes" (which output 1 if the inputs verify and output 0 otherwise) for proving knowledge of discrete logarithms or ciphertexts in an ElGamal cryptosystem:

1. *Discrete log ZKP* ($\mathcal{F}_{ZKP\text{-}DL}$)*:* For input (g, y) from the verifier, outputs 1, if the prover inputs x such that $g^x = y$, and 0 otherwise.

2. *Discrete-log equality ZKP* ($\mathcal{F}_{ZKP\text{-}DLE}$)*:* For input (g_1, y_1, g_2, y_2) from the verifier, outputs 1, if the prover inputs x such that $g_1^x = y_1 \wedge g_2^x = y_2$, and 0 otherwise.

3. *ElGamal re-encryption and re-randomization ZKP* ($\mathcal{F}_{ZKP\text{-}RR}$)*:* For input of generator g, public key y, input ciphertext pair (c_1, c_2), and output ciphertext pair (d_1, d_2) from the verifier, outputs 1, if the prover inputs re-encryption parameter r_1 and re-randomization parameter r_2 such that $((c_1 g^{r_1})^{r_2}, (c_2 y^{r_1})^{r_2}) = (d_1, d_2)$, and 0 otherwise.

4. *ElGamal re-encryption shuffle ZKP* ($\mathcal{F}_{ZKP\text{-}S}$)*:* For an input of shuffle input $((c_1^1, c_2^1), \ldots, (c_1^k, c_2^k))$ and output $((d_1^1, d_2^1), \ldots, (d_1^k, d_2^k))$ ciphertexts from the verifier, outputs 1, if the prover inputs permutation π and re-encryption parameters $(r_1, \ldots, r_k)$ such that $(c_1^i g^{r_i}, c_2^i y^{r_i}) = \left(d_1^{\pi(i)}, d_2^{\pi(i)}\right)$.

The details about the sigma protocols for these zero-knowledge proofs, and the method to obtain efficient proofs secure in the UC model can be found in the Appendix of the research work by Fenske et al. [67].

6.4.2 Initialization and Data Collection

Initialization. Each Data Party DP_i maintains a hash table T^i with b bins. Each bin has a value in $\mathbb{Z}_q$. Let $\mathcal{H}$ be the hash function used and assume that it can be modeled as a random function. If there are at most e inputs and we expect less than a fraction f of the inputs to experience a collision, then we set $b = e/f$ to obtain the desired accuracy. Each DP_i chooses values $r_k^{ij} \xleftarrow{\text{R}} \mathbb{Z}_q$, $1 \leq k \leq b$, $1 \leq j \leq m$, initializes each bin $T_k^i = -\sum_j r_k^{ij}$, sends each r_k^{ij} to Computation Party CP_j through $\mathcal{F}_{SC}$, and then removes the r_k^{ij} from memory. We can view this process as a secret sharing of a value $S_k^i = T_k^i + \sum_j r_k^{ij}$ between DP_i and the CPs, where S_k^i has an initial value of zero.

Data Collection. During data collection, DP_i records observed items in its hash table. For a given item z, DP_i chooses $r \xleftarrow{\text{R}} \mathbb{Z}_q$, computes $k = \mathcal{H}(z)$, and updates the kth bin as $T_k^i \leftarrow T_k^i + r$. This makes the secret-shared value S_k^i uniformly random in $\mathbb{Z}_q$. Such a value is non-zero with overwhelming probability (in q). So this process records the observation of an item, by interpreting a non-zero S_k^i to indicate an observed item. S^i contains the set of such observations. We note that secret sharing the bins between DP_i and the CPs, prevents the DP from storing any sensitive local state. Therefore, an adversary that gains access to a DP's state would just see a uniformly and independently random value in each bin.

Counter Forwarding. At the end of data collection, each DP_i sends its hash table to the CPs through additional secret sharing. That is, DP_i chooses values $s_k^{ij} \xleftarrow{\text{R}} \mathbb{Z}_q$, $1 \leq j \leq m-1$, $1 \leq k \leq b$, sets $s_k^{im} = T_k^i - \sum_{j=1}^{m-1} s_k^{ij}$, and sends each s_k^{ij} to Computation Party CP_j through $\mathcal{F}_{SC}$.

6.4.3 Aggregating Inputs

The CPs begin the secure computation process by generating an ElGamal keypair. A Computation Party CP_j chooses private key $x_j \xleftarrow{R} \mathbb{Z}_q$ and broadcasts public key $y_j = g^{x_j}$ to all other CPs. Note that CP_j uses $\mathcal{F}_{BC}$ for the current broadcast and all later ones, and if any of these broadcasts aborts, CP_j aborts. CP_j uses $\mathcal{F}_{\text{ZKP-DL}}$ to prove knowledge of x_j with each receiving Computation Party CP_i ($i \neq j$) as the verifier on y_j. If any proof fails to verify at CP_i (*i.e.*, $\mathcal{F}_{\text{ZKP-DL}}$ outputs 0 to CP_i), then CP_i aborts. A verifier also aborts on the failed verification of any future proof, and so we will not explicitly state this again. Each CP computes the joint public key as $y = \prod_j y_j$.

The CPs aggregate the hash tables by adding together all the shares, that they received from every DP for a given bin. That is, CP_j computes an aggregate table A^j where the kth bin contains $A^j_k = \sum_{i=1}^{d} r^{ij}_k + s^{ij}_k$. Thus each A^j_k is a share of a value $A_k = \sum_{j=1}^{m} A^j_k$, which is random if any of the DPs observed an item z such that $\mathcal{H}(z) = k$ and is zero otherwise. Therefore, under the interpretation that a non-zero value indicates the presence of an item in the table, A_k represents whether or not some DP recorded an observation in bin k (except with negligible probability in q), and the table A contains the union of the sets S^i produced by the DPs during data collection.

The CPs prepare these aggregate tables for input into an ElGamal shuffle. CP_j chooses $r \xleftarrow{R} \mathbb{Z}_q$, encodes the kth value as $g^{A^j_k}$, and encrypts it using public key y to produce ciphertext pair $c^j_k = (g^r, y^r g^{A^j_k})$. We place A^j_k in the exponent to take advantage of the ElGamal multiplicative homomorphism as an additive operation on the A^j_k values. Therefore, no discrete-log operation will be later needed to recover

the desired plaintext values, as we need to distinguish only between zero and non-zero exponent values.

CP$_j$ then broadcasts c_k^j to all other CPs. Each CP computes the encrypted ElGamal shuffle inputs as $c_k = \prod_j c_k^j$. Due to the ElGamal multiplicative homomorphism, c_k is an encryption of g^{A_k}, and thus the value c_k represents an encryption of the union of the kth bin values recorded by the DPs. CP$_j$ uses $\mathcal{F}_{ZKP\text{-}DL}$ to prove knowledge of r with each receiving Computation Party CP$_i$ ($i \neq j$) as the verifier on the first component of c_k^j (*i.e.*, c_{k1}^j), which implies knowledge of the encrypted value $g^{A_k^j}$.

6.4.4 Noise Generation

The CPs collectively and securely generate noise inputs to provide differential privacy to the output. As discussed in §3.2, (ϵ, δ)-differential privacy can be provided to counting queries by sampling n bits uniformly at random, where n is the smallest value satisfying Eq. 3.2. The CPs generate such bits using verifiable ElGamal re-encryption shuffles so that the resulting values are encrypted and can later be shuffled along with the encrypted inputs c_k.

The CPs run n ElGamal shuffle sequences in parallel – one for each noise bit. Each shuffle sequence has two input ciphertext pairs, $c_0^0 = (g^0, y^0 g^0)$ and $c_1^0 = (g^0, y^0 g^1)$, encoding '0' and '1' bit, respectively. Note that the randomness for the encryption is fixed as '0' so that all CPs know that the shuffle inputs are correctly formed. Let $c^0 = (c_0^0, c_1^0)$. Then each CP$_i$, in sequence from $i = 1$ to m, performs the following actions:

1. Re-encrypts c_0^{i-1} as c'_0 and c_1^{i-1} as c'_1.

2. Chooses $\beta \xleftarrow{R} \{0,1\}$, permutes the re-encryptions as $c^i = (c'_\beta, c'_{1-\beta})$, and broadcasts c^i to all other CPs.

3. Uses $\mathcal{F}_{ZKP\text{-}S}$ to prove that the shuffle was performed correctly, with each receiving Computation Party CP_j $(i \neq j)$ in parallel as the verifier on shuffle input c^{i-1} and output c^i.

From the final output c^m of the kth parallel noise shuffle, where $1 \leq k \leq n$, we take the first element c_0^m to be the kth noise bit, which we denote c_{b+k}.

6.4.5 Shuffling, Re-Randomization, Decryption, and Aggregation

Counter Shuffling. At this point, we have produced $v = b + n$ values encoded as ElGamal ciphertexts c_k, where the first b values contain the aggregated bins and the last n values contain the noise. To hide the values of specific bins and noise bins, the CPs shuffle these values before decryption. Let $c^{1,0} = (c_1, \ldots, c_v)$. To shuffle $c^{1,0}$, each CP_i, in sequence from $i = 1$ to m, performs the following actions:

1. Re-encrypts each ciphertext in $c^{1,i-1}$ to produce c'.

2. Chooses a random permutation π, permutes c' as $c^{1,i} = \left(c'_{\pi(1)}, \ldots, c'_{\pi(v)}\right)$, and broadcasts $c^{1,i}$ to all other CPs.

3. Uses $\mathcal{F}_{ZKP\text{-}S}$ to prove that the shuffle was performed correctly, with each receiving Computation Party CP_j $(i \neq j)$ in parallel as the verifier on shuffle input $c^{1,i-1}$ and output $c^{1,i}$.

Counter Re-Randomization. Next, we re-encrypt and then re-randomize the plaintexts of the final shuffle output, $c^{1,m}$. The re-randomization ensures that the encrypted values are each either g^0 or uniformly random in $G \backslash \{g^0\}$ and thus reveals

only one bit of information. To re-randomize, let $c^{2,0} = c^{1,m}$, and then each CP_i, in sequence from $i = 1$ to m, performs the following actions:

1. Re-encrypts each ciphertext in $c^{2,i-1}$ to produce c', re-randomizes each ciphertext in c' to produce $c^{2,i}$, and broadcasts $c^{2,i}$ to all other CPs.

2. Uses $\mathcal{F}_{\textit{ZKP-RR}}$ to prove that the re-encryption and re-randomization was performed correctly, with each receiving Computation Party CP_j ($i \neq j$) in parallel as the verifier on re-encryption and re-randomization input $c_k^{2,i-1}$ and output $c_k^{2,i}$, for $1 \leq k \leq v$. Each CP_j ($i \neq j$) also verifies that, for all k, the ciphertext $(c_1, c_2) = c_k^{2,i}$ is such that $c_1 \neq g^0$ and aborts if not. This check ensures that the re-randomization parameter was non-zero.

Counter Decryption. Finally, the CPs decrypt the result. Let $c^{3,0} = c^{2,m}$. Each CP_i, in sequence from $i = 1$ to m, performs the following actions:

1. Decrypts each ciphertext in $c^{3,i-1}$ using key x_i to produce $c^{3,i}$.

2. For each $1 \leq k \leq v$, let $(c_1, c_2) = c_k^{3,i-1}$ and $(c_3, c_4) = c_k^{3,i}$, uses $\mathcal{F}_{\textit{ZKP-DLE}}$ with each receiving Computation Party CP_j ($i \neq j$) in parallel as the verifier on $(g, y_i, c_1, c_2/c_4)$. Each CP_j also verifies that $c_3 = c_1$ and aborts if not. These steps prove that the decryption was performed correctly.

Aggregation. Let p_i, $1 \leq i \leq v$, be the second component of $c_i^{3,m}$, which is a plaintext value, and let b_i be 0 if $p_i = g^0$ and be 1 otherwise. Each CP produces the output value $z = \sum_{i=1}^{v} b_i - n/2$, where the $-n/2$ term corrects for the expected amount of added noise.

6.4.6 Optimizations

There are a number of practical optmizations that can be made to the base PSC protocol. They are as follows:

- During initialization, instead of sending b random values to the CPs, the DPs can just send short random seeds. Then both the parties can use a seeded pseudorandom generator to derive the b random values.
- The counter re-randomization and decryption phases can be combined, so that the added rounds in the final sequential decryption can be eliminated. This requires a zero-knowledge proof for the combined re-encryption, re-randomization, and decryption operations.
- The zero-knowledge proofs can be made non-interactive via the Fiat-Shamir heuristic [68].
- The noise generation phase can be done in advance, for example while the DPs are collecting data, as it does not require any inputs from the DPs.

6.4.7 Security and Privacy

Although the base PSC protocol does not provide full accountability and can easily be disrupted by malicious parties (see §6.3), it still provides strong security and privacy guarantees. It is provably secure in the Universally Composable framework [67], a strong notion of security that allows PSC to serve as a component of a larger UC-secure system without losing its security properties. The security of PSC in the UC model also directly implies its security in weaker models, such as the standalone model.

We note that, security against adaptive corruptions is often difficult to prove with efficient protocols. However, we need adaptive security to guarantee *forward privacy*, that is nothing can be learnt about past inputs by corrupting a DP. Therefore, we allow adaptive corruptions against the DPs only. The base PSC protocol is proved secure against adaptive DP corruptions and static CP corruptions as long as at least one CP is honest (for details, see Fenske et al. [67]).

Moreover, the base PSC protocol produces differentially private output, which looks nearly the same to the adversary, irrespective of whether a single item was observed by some DP or not.

6.4.8 Lack of Accountability

The base PSC protocol does not provide full accountability against malicious parties. Even a single malicious DP (or CP) can cause DoS for an epoch by not taking part in the statistics collection (or aggregate computation) process. This can be a serious threat at least in case of volunteer-operated anonymity networks (such as Tor), an adversary can easily operate a malicious node.

Another serious threat is that, a malicious CP can set any counter of its choice to a "1" (by multiplying its share of the counter with a random value), without detection. This threat can be mitigated by letting the DPs encrypt their inputs (with the ElGamal joint public key y of the CPs) and broadcast them to all the CPs directly. However, this would allow malicious DPs to equivocate by sending different set of encrypted inputs to different subset of CPs, without detection. Therefore, in order to provide accountability against malicious DPs, the DPs must sign and broadcast their encrypted inputs to all CPs. Though this solution seems to provide full accountability against malicious parties at first blush, it uncovers other major problems such as joint public key y distribution to DPs (for encrypting DP inputs),

DP key generation (for signing DP inputs), protection against malicious DPs doing a target re-encryption attack during input submission (for details refer to §6.5.3), and lack of an efficient approach for CPs to remove a subset of equivocating DPs (to provide DoS resistance).

6.5 Enhanced PSC Protocol with Accountability

We enhance the base PSC protocol to provide full accountability against malicious parties and address all the issues mentioned in §6.4.8. We enhance the base PSC protocol (described in §6.4) by describing additional preliminaries and the different protocol phases in detail.

6.5.1 Preliminaries

In addition to ($\mathcal{F}_{SC}$) (described in §6.4.1), we assume PSC has access to the following cryptographic primitive as a "black-box" (to provide ideal functionality):

- *Consensus broadcast-with-blame functionality* ($\mathcal{F}_{BB}$)*:* Guarantees that all honest parties agree on the same message or terminate by blaming the broadcaster. Every honest party agrees on the blame message as well.

We also assume PSC has access to the following zero-knowledge proof of knowledge as a "black-box" (which outputs 1 if the inputs verify and outputs 0 otherwise), in addition to $\mathcal{F}_{ZKP\text{-}DL}$, $\mathcal{F}_{ZKP\text{-}DLE}$, and $\mathcal{F}_{ZKP\text{-}S}$ (described in §6.4.1):

- *ElGamal re-randomization and decryption ZKP* ($\mathcal{F}_{ZKP\text{-}RRD}$)*:* For input of generator g, public key $y_j = g^{x_j}$, partial public key $y = \prod_i y_i$, input ciphertext pair (c_1, c_2), and output ciphertext pair (d_1, d_2) from the verifier, outputs 1, if the

prover inputs re-encryption parameter r_1 and re-randomization parameter r_2 such that $((c_1 g^{r_1})^{r_2}, ((c_2 y^{r_1})^{r_2}) {c_1}^{-x_j}) = (d_1, d_2)$[1], and 0 otherwise.

6.5.2 Blame

A major change from the base PSC protocol is that each Computation Party aborts only with a blame message (blaming a misbehaving party). Every honest Computation Party agrees on the blame message, which gives the required accountability guarantee.

Each Computation Party now uses $\mathcal{F}_{BB}$ (instead of ($\mathcal{F}_{BC}$)) for all broadcasts among the CPs. If any of the broadcasts aborts with a blame message, then the CPs check the blame message: if a CP is blamed, all Computation Parties terminate and output the blame message. However, if a DP is blamed, then the CPs drop that DP's input and continue.

Similarly, if any of the ZKPs' (described in §6.5.1) proof fail to verify, then the verifier aborts and blames the prover. We will not explicitly state "abort and blame" again for all future broadcasts and proofs.

6.5.3 Protocol Description

CP Key Generation. The CPs begin the PSC protocol by generating an ElGamal keypair (as described in §6.4.2). A Computation Party CP_j chooses private key $x_j \xleftarrow{R} \mathbb{Z}_q$ and broadcasts public key $y_j = g^{x_j}$ to all other CPs. CP_j uses $\mathcal{F}_{ZKP\text{-}DL}$ to prove the knowledge of x_j. Each CP computes the joint public key as $y = \prod_j y_j$.

Joint Public Key Distribution. Each Computation Party CP_i signs and sends the computed joint public key y to each Data Party DP_i. Each Data Party DP_i then

[1] *i.e.*, (d_1, d_2) is a partial decryption of a re-encryption re-randomization of (c_1, c_2)

verifies if it has received the same copy of joint public key from all the CPs. If not, DP_i sends any two joint public keys (received from the CPs) that are not the same to all the CPs and terminates.

On receiving the joint public key from DP_i, each Computation Party CP_j verifies the received joint public key against its own copy of the joint public key, blames the Computation Party that signed the joint public key which does not verify, and terminates.

Initialization. Each Data Party DP_i maintains a hash table T^i with $b = e/f$ (see §6.4.2 for details) bins. Each bin value is an ElGamal ciphertext. Each DP_i initializes each bin with random ElGamal encryptions of unit element, using the joint public key y of the CPs: $T^i_k = (g^r, y^r)$, $1 \leq k \leq b$.

Data Collection. During data collection, DP_i records observed items in its hash table. For a given item z, DP_i chooses $r, s \xleftarrow{R} \mathbb{Z}_q$, computes $k = \mathcal{H}(z)$. Let kth bin be $T^i_k = (R, C)$ (an ElGamal ciphertext of unknown value). DP_i updates the kth bin by replacing it with a random ElGamal ciphertext: $T^i_k = (g^s, y^s g^r)$. Such a value is an encryption of a non-unit element with overwhelming probability (in q). So just like in the base PSC protocol (see §6.4.2), this process records the observation of an item, by interpreting a non-unit ElGamal encryption to indicate an observed item. We note that the bins are encrypted with the joint public key y of the CPs, and hence a DP cannot decrypt its own counters. Therefore, an adversary that gains access to a DP's counters would just see a uniformly and independently random ElGamal ciphertext in each bin.

Counter Forwarding. At the end of data collection, each DP_i first broadcasts only a commitment of its hash table to the CPs. This ensures that a malicious DP cannot do a target attack by submitting re-encryptions of some other DP's target bin as its input for every bin. Finally, in the subsequent round, each DP broadcasts the

hash table itself. On receiving the hash table from every Data Party, each Computation Party CP_j verifies the hash table with the corresponding commitment sent in the previous round, blames the Data Parties whose commitments do not verify, and terminates.

6.5.4 Aggregating Inputs

The CPs aggregate the hash tables by multiplying together all the shares, that they received from every DP for a given bin. That is, CP_j computes an aggregate table A where the kth bin contains $c_k = \prod_i T_k^i$. Due to the ElGamal multiplicative homomorphism, c_k represents an encryption of the union of the kth bin values recorded by the DPs.

6.5.5 Noise Generation

The CPs collectively and securely generate noise inputs to provide differential privacy to the output. As discussed in §6.4.4, the CPs generate n noise bits using verifiable ElGamal re-encryption shuffles so that the resulting values are ElGamal encrypted and can later be shuffled along with the encrypted inputs c_k. The CPs blame the sender if $\mathcal{F}_{\text{ZKP-S}}$ fails on any shuffle input and output. We denote the kth noise bit by c_{b+k}, where $1 \leq k \leq n$.

6.5.6 Shuffling, Re-Randomization, Decryption, and Aggregation

Counter Shuffling. At this point, we have produced $v = b + n$ values encoded as ElGamal ciphertexts c_k, where the first b values contain the aggregated bins and the last n values contain the noise. To hide the values of specific bins and noise bins, the CPs shuffle these values before decryption as described in §6.4.5. Here again,

the CPs blame the sender if $\mathcal{F}_{ZKP\text{-}S}$ fails on any shuffle input and output. Let the final shuffle output be $c^{1,m}$.

Counter Re-Randomization and Decryption. As discussed in §6.4.6, this step combines the counter re-randomization and decryption steps described in §6.4.5. Let $c^{2,0} = c^{1,m}$. Each CP$_i$, in sequence from $i = 1$ to m, performs the following actions:

1. Re-encrypts and re-randomizes each ciphertext in $c^{2,i-1}$ to produce c', decrypts each ciphertext in c' using key x_i to produce $c^{2,i}$, and broadcasts $c^{2,i}$ to all other CPs.

2. Uses ($\mathcal{F}_{ZKP\text{-}RRD}$) to prove that the combined re-randomization and decryption was performed correctly, with each receiving Computation Party CP$_j$ ($i \neq j$) in parallel as the verifier on re-encryption and re-randomization input $c_k^{2,i-1}$ and output $c_k^{2,i}$, for $1 \leq k \leq v$. To ensure that the re-randomization parameter is non-zero, each CP$_j$ ($i \neq j$) also verifies that, for all k, the ciphertext $(c_1, c_2) = c_k^{2,i}$ is such that $c_1 \neq g^0$ and aborts if not (as described in §6.4.5).

Aggregation. This step is exactly similar to the aggregation step described in §6.4.6. Let p_i, $1 \leq i \leq v$, be the second component of $c_i^{2,m}$, which is a plaintext value, and let b_i be 0 if $p_i = g^0$ and be 1 otherwise. Each CP produces the output value $z = \sum_{i=1}^{v} b_i - n/2$, where the $-n/2$ term corrects for the expected amount of added noise.

6.5.7 Security and Performance Improvements

This version of the PSC protocol provides full accountability against malicious parties, in addition to all the security and privacy guarantees provided by the base PSC protocol (for details refer to §6.4.7). The accountability property guarantees that all

honest parties terminate successfully or by blaming some malicious (or misbehaving party). We use a modified version of the Dolev-Strong consensus protocol [55] to achieve the ($\mathcal{F}_{BB}$) functionality described in §6.5.1. The full-version of the proof is in preparation [66].

In order for PSC to be secure in a dishonest majority adversarial setting, the Dolev-Strong consensus protocol [55] requires a communication cost that is quadratic in the number of Computation Parties for every CP-CP communication. Therefore, to improve the efficiency of our implementation, we instead use an *echo* broadcast protocol (which is the first two rounds of the Dolev-Strong consensus protocol) to instantiate ($\mathcal{F}_{BB}$). The *echo* broadcast protocol provides full accountability as long as all parties send "valid" (*i.e.*, properly formatted) messages. However, since all the rounds of the Dolev-Strong protocol is required to provide consensus, the implementation does not guarantee termination in the dishonest setting. Some honest parties may terminate by blaming some malicious party, while other honest parties may wait on the terminated parties forever.

The security of PSC in the UC model also directly implies its security in weaker models, such as the standalone model. Therefore, in order to improve efficiency of our implementation furthermore, we accept a weaker security model and instantiate a functionality needed by PSC using a protocol that can only be proved in the weaker model (*e.g.*, using the Neff shuffle [110] for $\mathcal{F}_{ZKP\text{-}S}$).

6.6 Implementation and Evaluation

We constructed an implementation of PSC in Go to verify our protocol's correctness and to measure the system's computation and communication overheads. We run experiments over large synthetic datasets and measure our implementation's perfor-

mance. We describe the implementation details and design choices in §6.6.1, and then finally present our performance evaluation in §6.6.2.

6.6.1 Implementation

We built an implementation of the enhanced PSC in 2500 lines of Go using the DeDiS Advanced Crypto Library for Go package [2]. ElGamal encryption is implemented in the Edwards 25519 elliptic curve group [113] and the CPs use Neff's verifiable shuffle [110] to shuffle the ElGamal ciphertexts.

The CPs instantiate accountable broadcast functionality using a two-round echo-broadcast protocol with accountability (see §6.5.1). They use the Schnorr signature algorithm [122] over the Edwards 25519 elliptic curve for signing and SHA-256 for computing digests. We rely on TLS 1.2 to provide secure point-to-point communication.

We use "Biffle" in the DeDiS Advanced Crypto Library for shuffling the noise vectors during the noise generation phase. Biffle is a fast binary shuffle for two ciphertexts based on generic zero-knowledge proofs.

For zero-knowledge proofs, we use Schnorr proofs [122] for knowledge of discrete log of the Elgamal public key and blinding factors, the Chaum-Pedersen proof [39] for equality of discrete log of public key, and a modified generalization of the Chaum-Pedersen proof [110] for the combined re-encryption and decryption. Non-interactive versions of all these proofs are produced using the Fiat-Shamir heuristic [68].

A single DP program emulates all the DPs in our implementation. However, in a real deployment, the DPs would be distributed.

To encourage the use of PSC by privacy researchers and practitioners, we are releasing PSC as free, open-source software, available for download at http://safecounting.com.

6.6.2 Evaluation

We carried out experiments on 10 core Intel Xeon machines with 32GB to 64GB of RAM running Linux kernel 4.4.0-154-generic. Our implementation of PSC is currently single-threaded. Although the computational cost of PSC's noise generation is significant and may be done by the CPs before the inputs are received, we parallelize it so that it can be done on multicore machines after inputs are received. We use "parallel for" from the Golang par package [7] for this parallel noise shuffling.

We instantiate all CPs and DPs on our 10 core servers. Google Protocol Buffers [138] is used for serializing messages, which are communicated over TLS connections between PSC's parties. We use Go's default crypto/tls package to implement TLS.

Query and Dataset. We evaluate PSC by considering the query: what is the number of unique IPs as observed by the nodes in an anonymity network? Rather than store 2^{32} (or, for IPv6, 2^{128}) counters, we assume b counters (where $b \ll 2^{32}$) and map IP addresses to a $(\lg b)$-bit digest by considering the first $\lg b$ bits of a hash function; this results in some loss of accuracy due to collisions. For each experiment, we chose an integer uniformly at random from the range $[0, 30000]$. Then for each DP, we choose from its counters a random subset of that size to set to 1; the remaining counters are set to 0.

We note that the performance of PSC is independent of the number of unique IPs. Therefore, in our performance evaluation, we are interested in how the number of counters b affects the operation of PSC, rather than the number of unique IPs.

Table 6.1: Default values for PSC system parameters.

Parameter	Description	Default
b	number of counters	$300,000$
m	number of Computation Parties	5
d	number of Data Parties	30
ϵ	privacy parameter	0.3
δ	privacy parameter	10^{-12}

Experimental Setup. The default values for the number of bins b, the number of CPs m, the number of DPs d, ϵ, and δ are listed in Table 6.1.

We determine these default values by considering which values would be appropriate for an anonymity network such as Tor. Currently, Tor reports 2.7 million user connections (using simple statistical techniques) and over 3000 guard nodes [130]. Assuming that 1% of Tor guards deploy PSC, we have $d = 30$. Also, given this level of PSC deployment, we would expect a Tor guard running PSC to see approximately $30,000$ unique IPs.

To measure the aggregate with high accuracy, we limit hash-table collisions to at most a fraction f of inputs in expectation by using a hash table of size $1/f$ times the number of inputs. Therefore, for $f = 10\%$, we set (unless otherwise specified) $b = 300,000$ in all our experiments. We set $\epsilon = 0.3$ as this is currently recommended for safe measurements in anonymity networks such as Tor [80]. To limit to 10^{-6} the chance of a privacy "failure" affecting any of 10^6 users, we set δ to 10^{-12} [59]. We set these default values as system-wide parameters, unless otherwise indicated.

Accuracy. The trade-off between accuracy and privacy is governed by the choice of ϵ and δ. We try a range of values for ϵ between 0.3 (on the conservative side) to 7.5, keeping the number of bins at $b = 300,000$. We found that values below 0.3

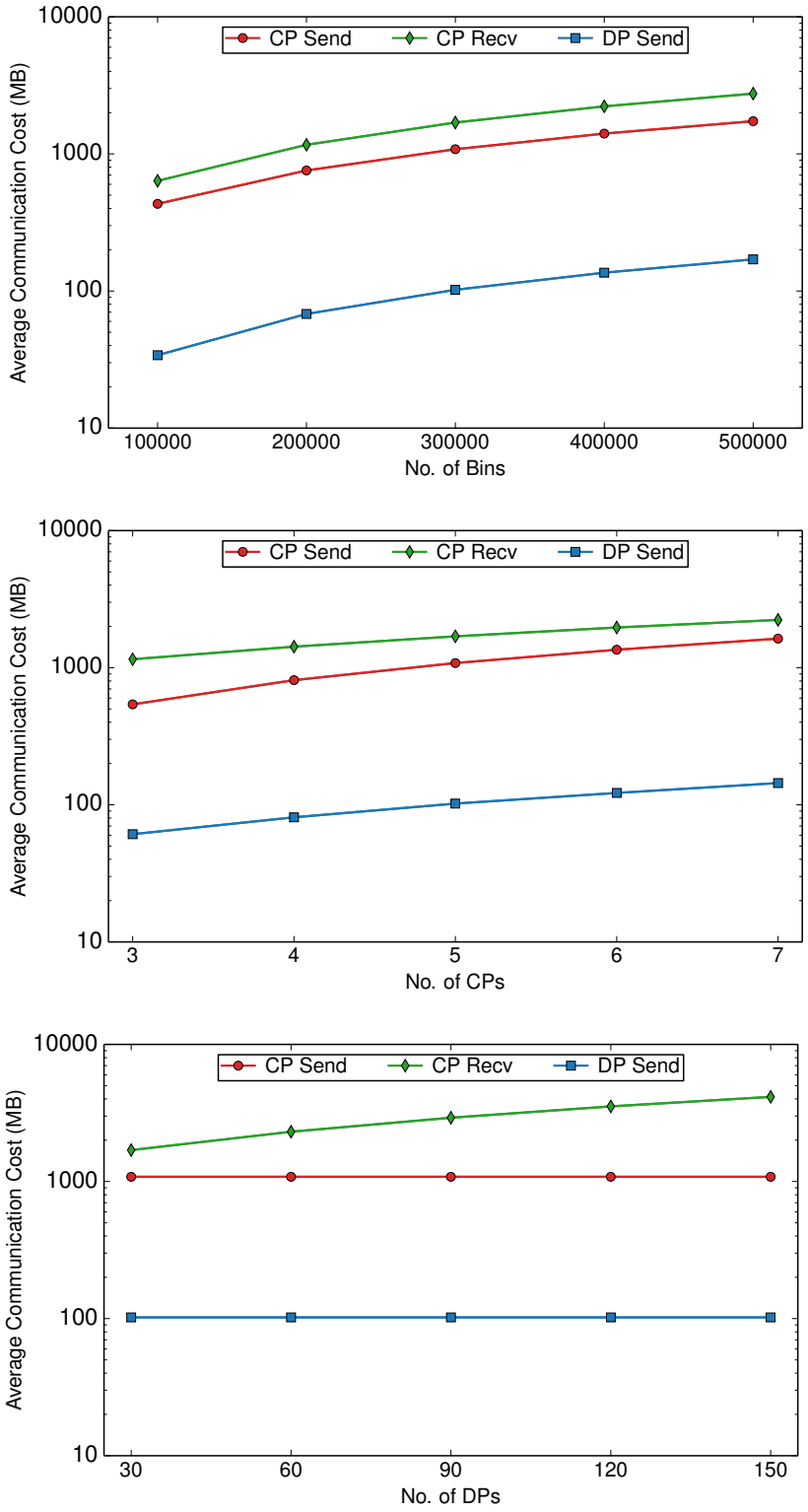

Figure 6.2: PSC communication cost varying the number of bins, CPs, and DPs. The communication cost incurred by the CPs and DPs varying the number of bins (*top*), the number of CPs (*middle*) and the number of DPs (*bottom*).

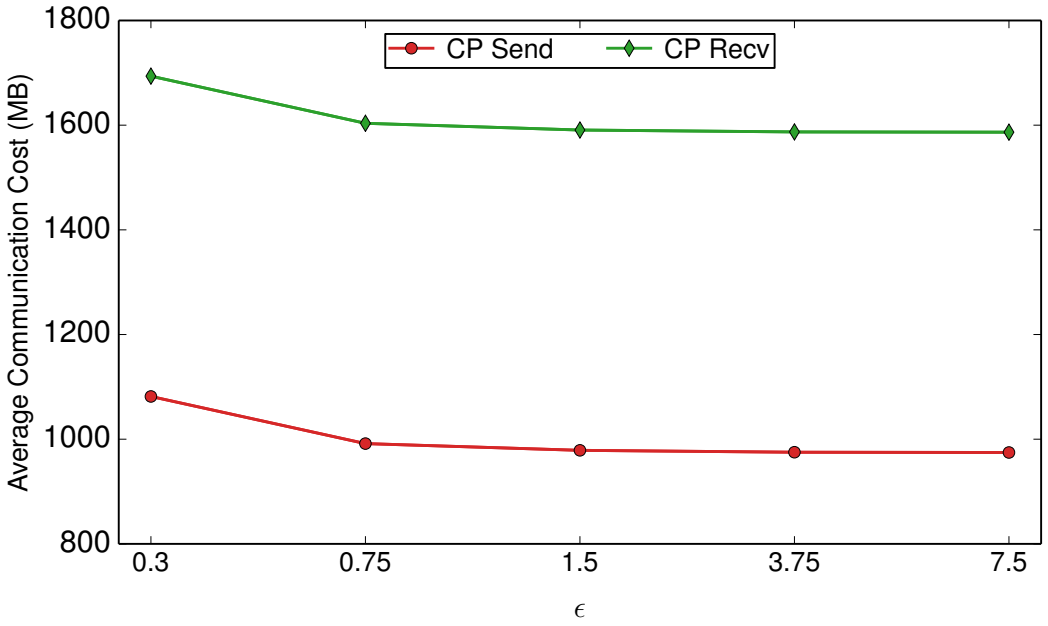

Figure 6.3: PSC communication cost as a function of ϵ. The communication cost incurred by the CPs varying ϵ.

produced too much noise and offered low utility. Values of ϵ greater than 7.5 would not provide a reasonable level of privacy.

The actual and the noisy aggregate values along with the standard deviation for the noisy aggregates (the noise follows a binomial distribution) for different values of ϵ is shown in Table 6.2. We observe that the standard deviation of the noisy value is at most 70.96. Therefore, the noisy aggregates are very close to the actual aggregates, as expected. In summary, PSC gives highly accurate results for the desired privacy level.

Communication Cost. PSC incurs communication overhead by transmitting ElGamal encrypted counters between the DPs and CPs and among the CPs themselves. To be practical, a statistics gathering system should impose a low communication overhead for the DPs, which can have limited available bandwidth. However,

Table 6.2: Standard deviation of the noisy PSC output as a function of ϵ.

Epsilon	Actual Aggregate	Noisy Aggregate	Standard Deviation
0.3	8927	8897	70.96
0.75	8927	8882	28.38
1.5	8927	8918	14.19
3.75	8927	8921	5.67
7.5	8927	8925	2.84

Actual, noisy aggregate, and standard deviation for various values of ϵ.

we envision the CPs to be well-resourced dedicated servers that can sustain at least moderate communication costs.

We explore PSC's communication costs by varying the number of bins b, the number of CPs m, the number of DPs d, ϵ, and δ. The average communication costs per CP and DP are plotted in Figure 6.2 and Figure 6.3. We omit error bars as the variance in the communication cost incurred among the CPs and DPs were negligible as the protocol has deterministic communication cost.

We first consider how the number of bins influences communication cost. We run PSC, varying b from 100,000 to 500,000, and plot the results in Figure 6.2. The values of the bins (*i.e.*, `0` or `1`) do not affect the communication cost of the DPs, as a DP transmits an ElGamal encrypted value for either `0` or `1`. For up to 300,000 bins, the outbound communication cost for each DP is fairly modest. For example, if PSC is run once an hour, then the communication cost is approximately 102 MB/hr (28.33 KBps). We find that the inbound communication cost for each

DP is fairly constant (1.82KB). This is because, the DPs only receive the the signed ElGamal joint public key from the CPs, irrespective of the number of bins.

The communication costs are more significant for the CPs, which we envision are dedicated machines for PSC. With 300,000 bins, each CP requires a bandwidth of 1.08 GB for outbound and 1.69 GB for inbound (approximately 300.46 KBps for outbound and 470.46 KBps for inbound if executed once per hour). The difference between the CP outbound and CP inbound communication is large (and approximately equal to the sum of outbound communication cost across all DPs) as each CP receives b ElGamal encrypted counters from all DPs, whereas just sends signed ElGamal joint public key (*i.e.*, a single group element) to all DPs.

We next consider how m, the number of CPs, affects the communication cost. We vary m from three to seven and plot the results in Figure 6.2. The outbound communication cost for the DPs increases at a much faster pace than the inbound communication cost. This is because the DPs send b (= 300,000) ElGamal encrypted counters to each CP. Still, for up to six CPs, the outbound communication cost for each DP is fairly modest – approximately 122.40MB (or 34 KBps, if run every hour).

The communication costs increase at a much faster pace for the CPs as each CP echo broadcasts proofs and ciphertexts to the other CPs. Therefore, for up to four CPs, each CP requires a modest bandwidth of 811.24 MB for outbound and 1.42 GB for inbound (if run hourly, approximately 225.34 KBps for outbound and 394.44 KBps for inbound). Here again, we can see that the difference between CP outbound and CP inbound communication cost is significant due to the asymmetry in the CP-DP communication.

We rerun PSC with different numbers of DPs. Figure 6.2 shows that varying the number of DPs has little effect on the average communication costs for the DPs and the average outbound communication cost for the CPs because the number of

encrypted counters transmitted by each DP and the number of ElGamal encrypted counters transmitted between the CPs remain the same across these experiments. However, there is a small linear increase in the inbound communication cost of the CPs as each CP receives b ElGamal encrypted counters from each DP. The variation in the inbound communication cost per DP is approximately 102MB.

Lastly, we run PSC with different values of ϵ to determine how the choice of privacy parameter affects the communication costs. Figure 6.3 shows that the average communication costs for the CPs decrease when ϵ is increased. The communication costs for the CPs are reasonable even for a low value of $\epsilon = 0.3$. On average, each CP requires a bandwidth of at most 1.08 GB for outbound and 1.69 GB for inbound (300 KBps for outbound and 469.44 KBps for inbound, if performed once an hour).

In summary, we find that PSC incurs reasonable communication overhead: the costs to DPs are fairly modest, and, while slightly higher for CPs, they remain practical.

Overall Runtime and Computation Cost. To understand the computation costs of PSC, we perform a series of microbenchmarks. We focus on evaluation of the CPs, as the DPs simply multiply ElGamal ciphertexts whenever they observe a client connection. Also the time taken to compute the hash for commitments is negligible. That is, the computation overhead of a DP is negligible.

We observe that the time taken for each operation of the CPs (including CP public key generation, joint public key computation and signing, aggregate computation, parallel noise generation, verifiable shuffling, and re-randomization and decryption) takes less than 6% of the total time taken in a run of the protocol. This is because the time required to transfer messages via echo broadcast overwhelms the time required for computation.

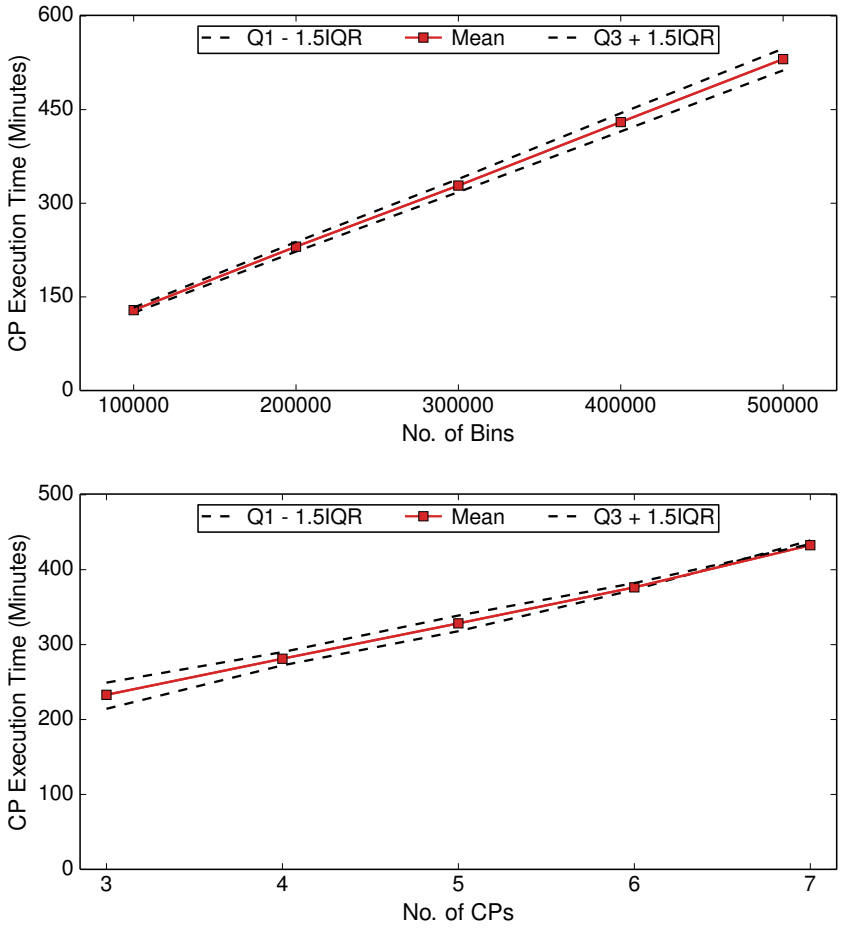

Figure 6.4: PSC computation cost as a function of the number of bins and CPs. The average execution time as a function of the number of bins (*top*) and the number of CPs (*bottom*). The dashed-lines show the range within 1.5xIQR from the lower and upper quartiles.

Figure 6.4 shows the overall running time (including the time required for network communication) as a function of the number of bins b and the number of CPs m. We first consider how number of bins b affects the computation cost (note that more data must be communicated via echo broadcast as b increases). The overall runtime is fairly moderate. It takes approximately 8 hr 50 min even for an experiment with 500,000 bins.

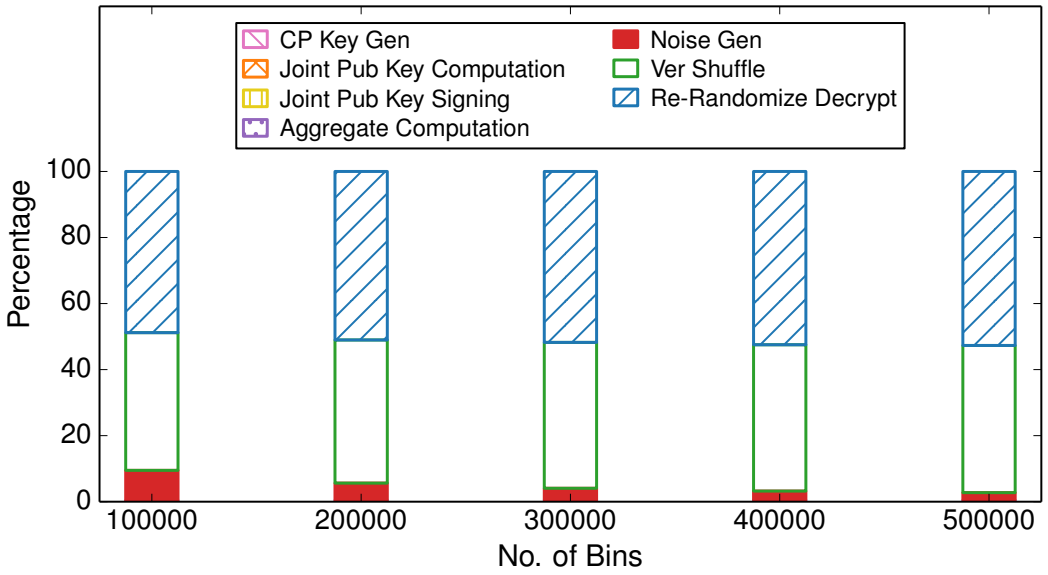

Figure 6.5: PSC computation cost per operation. The computation cost incurred by the CPs for different operations, varying the number of bins.

We next consider how the number of CPs m affects the computation cost. The computation cost increases with the CPs at a slower pace than with the bins. Even up to five CPs, the average computation cost for each CP is fairly modest – approximately 5 hr 29 min.

To better understand the computational overhead of PSC, we measure the computation time as we vary the number of bins b. Figure 6.5 shows the distribution of processing overheads for the CPs for different operations, which account for less than 6% of the total execution time.

As we can see, only the verifiable shuffle and counter re-randomization decryption phases account for more than 90% of the CP execution time. However, since, the processing overhead is only 6% of the total execution time, the implementation is fairly optimized.

6.7 Related Cryptographic Protocols

In this section, we briefly review some of the cryptographic protocols which are related to the design of PSC.

Brandt suggests a protocol [33] very similar to the aggregate-shuffle-rerandomize-decrypt scheme our aggregators execute. However, our construction differs in a few crucial parts: we include separate data collecting parties (or relays) to provide the input, provide resistance to compulsion attacks (so that the relays cannot release sensitive information under pressure), and limit as much as possible their computational work. Moreover, in our protocol the aggregating parties jointly generate noise to satisfy a differential privacy guarantee. Yet another difference is that PSC can be proved secure in the UC model. The proof is in preparation [66] (the base PSC protocol is proved secure in the UC model and the details can be found in the appendix of the research work by Fenske et al.). Beyond these modifications, the bulk of our contribution is a specific application for the general theoretical protocol (making measurements in privacy-preserving systems like Tor), and a functional implementation of the protocol alongside empirical data measuring computation and communication costs gathered through experiments.

Several protocols to securely compute set-union cardinality (or the related problem of set-intersection cardinality) have been proposed [50, 61, 88, 136]. However, none of these provides all of the security properties that we desire for distributed measurement in a highly-adversarial setting: malicious security against a dishonest majority, forward privacy, and differentially-private output. Similar protocols have been designed to securely compute set union [69, 79], but these protocols output every member of the union and not just its cardinality. General secure multiparty computation (MPC) protocols can realize any functionality including set-union cardinal-

ity, even in the UC model [26, 36], and recent advances in efficient MPC have been made in the multi-party, dishonest-majority setting that we require [48, 93, 98]. However, such protocols make use of a relatively expensive "offline" phase, while we intend to allow for measurements that are run on a continuous basis. We also consider a major advantage of PSC to be that it is conceptually straightforward and relies on a few well-understood tools.

6.8 Summary

In this chapter, we present PSC, a protocol for counting the number of unique items distributed across the system's nodes, without exposing any information other than the cardinality. PSC computes private set-union cardinality using a combination of verifiable shuffles and differential privacy techniques. It is provably secure in the Universally Composable framework against an active adversary that compromises all but one of the aggregation parties.

We demonstrate using a proof-of-concept implementation that PSC is practical for real-world deployments. In particular, for reasonable deployment sizes, PSC runs at timescales smaller than the typical measurement period and thus is suitable for privacy-preserving measurements in distributed systems, such as anonymity networks.

In the next chapter, we describe significant enhancements for PSC to make it suitable for deployment in the live Tor network, and explore Tor user behavior that has never been measured before.

Chapter 7

Understanding Tor Usage with PSC

In this chapter, we significantly enhance PSC to support the safe exploration of Tor user behavior. We have integrated the improvements described in this chapter into the open-source PSC implementation [100]. Using the enhanced version of PSC, we conduct a comprehensive measurement study of Tor that covers three major aspects of its usage: how many unique client IPs connect to Tor and from where they connect; how many unique destinations do Tor users visit; and how many unique onion services exist. To better understand who uses Tor and from where they access the network, we also analyze a number of client measurement results obtained using an enhanced PrivCount deployment.

As perhaps our most significant result, we find that between 6 to 11 million unique client IP addresses connect to Tor daily, a factor of three to five more than previously believed. Additionally, we are the first to measure the client IP churn in Tor, and we observe that the number of new unique client IPs connecting to Tor decreases over time.

Our findings both reinforce existing beliefs about the Tor network (*e.g.*, the number of unique onion addresses published) and elucidate many aspects of the Tor network that have previously not been explored. In some instances, our measure-

ments – which are based on *direct* observations of user behavior on Tor – suggest that existing heuristically-driven estimates of Tor's usage (including the number of Tor users) are highly inaccurate.

We run 16 Tor relays that contribute approximately 3.5% of Tor's bandwidth capacity, and deploy PSC across these relays to safely collect measurements of the live Tor network. We extend previous statistical methodology to enable unique-counting on Tor, and extend privacy definitions ("action bounds") to cover new types of user activity. We also describe methods for inferring whole-network statistics given local observations at our relays.

Limiting the risk to Tor's users was paramount in both the design and implementation of all these measurements, and a significant contribution is the methodology for choosing system parameters to guarantee adequate levels of protection. We discuss the precautions we took to ensure user safety, argue for the ethical validity of our study, and describe our experiences with institutional ethics boards and an independent safety review board.

7.1 Background

We begin this chapter by presenting a brief overview of Tor and reviewing the PrivCount measurement tool.

7.1.1 Background on Tor

Tor provides anonymous TCP connections through source-routed paths that originate at the Tor client (*i.e.*, the user) and traverse through (usually) three relays (refer Figure 7.1). Traffic enters the Tor network through *guard relays*. Middle relays carry traffic from guard relays to *exit relays*, where the traffic exits the anonymity

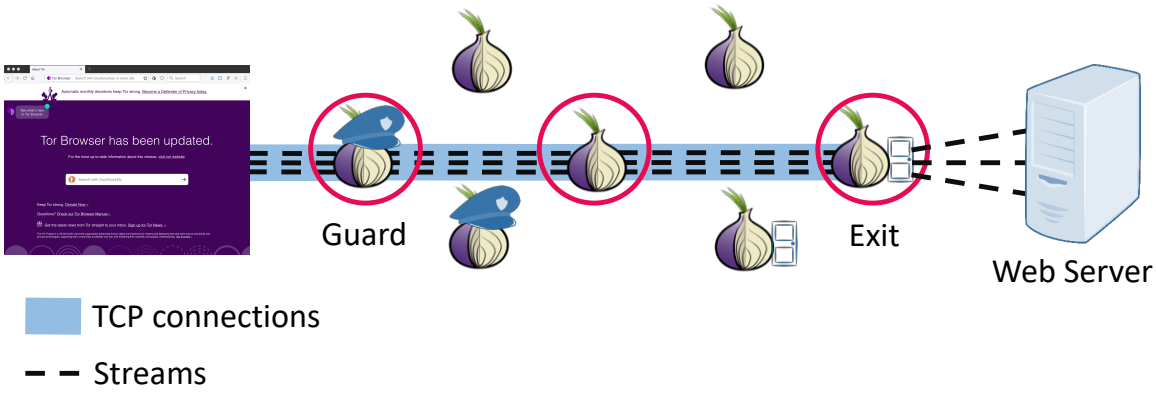

Figure 7.1: Tor overview. An overview of the Tor network highlighting Tor circuit (*i.e.*, anonymous path through the circled Tor relays) and streams (dashed lines).

network. These anonymous paths through Tor relays are called *circuits*. The unit of transport in Tor circuits is the *cell*: Tor cells carry encrypted routing information and 498 bytes of data [51]. The encryption of application-level headers and payloads makes it difficult for an adversary to perform traffic analysis on circuits and learn the endpoints or content of communication.

Tor has a three-tiered communication architecture. As shown in Figure 7.1, a single circuit may carry multiple *streams*. A Tor stream is a logical connection between the client and a single destination (*e.g.*, a webserver), and is roughly analogous to a standard TCP connection. A request to retrieve a webpage may produce several streams (*e.g.*, to `example.com` and `ads.example.com`) that are multiplexed over a single circuit.

Finally, to reduce latency (by avoiding TCP slow-start at multiple locations), Tor maintains longstanding TLS-encrypted *connections* (shown as a blue solid line in Figure 7.1) between relays that are adjacent on some Tor circuit. Communication belonging to different circuits that share a common hop between two relays are multiplexed through a single connection. Similarly, a user's Tor client maintains a single

connection to each of its guards, through which multiple circuits (and streams) may be formed.

Tor clients periodically download "directory updates" that describe the consensus view of the network. Clients send all user data through one guard by default, but directory updates are obtained through three guards by default. Tor clients avoiding censorship may connect to the network through *bridges* [104], which are guards whose identities are not public and are disseminated by Tor to users requesting them.

In summary, circuits carry one or more streams from a client through a common set of Tor relays (usually three); and multiple circuits belonging to potentially different Tor users may be carried over a TCP connection that exists between two relays.

Onion Services. Tor allows services to hide their locations through the use of *onion services* [129]. As a special (but common) case, when the onion service corresponds to a hidden website, we refer to it as an *onionsite*.

To operate an onion service, the operator selects six relays to serve as *introduction points*. It then (1) constructs an *onion service descriptor* that contains its public key and the identities of the chosen introduction points, and (2) forms Tor circuits to each of the introduction points. The descriptor is then stored on six or eight relays [11, 12] (depending on the version of Tor) using a distributed hash table (DHT) [84]. These relays are called onion service directories (for historical reasons, they are also called "hidden service directories" or "HSDirs"). The DHT is indexed by the descriptor ID, which is derived from the public key. The address of the service is a domain with an `.onion` suffix, which is derived from the public key.

When a user wishes to access the onion service, it queries the DHT to obtain the service's onion descriptor, which includes the identities of its introduction points. It also chooses a relay to serve as a *rendezvous point* (RP), and constructs a circuit to

this RP. Using the public key in the onion descriptor, the user encrypts its choice of the RP and sends it via a Tor circuit to one of the onion service's introduction points. This choice is then forwarded to the onion service, using the existing circuit between the introduction point and the service. Finally, the onion service constructs a circuit to the RP, through which it can communicate with the user.

Importantly, Tor conceals the network locations of both the user and the onion service. The communications between the user and the introduction point, the user and the RP, the onion service and the introduction point, and the onion service and the RP are *all* carried over Tor circuits.

7.1.2 PrivCount

Recall from §2.2 that PrivCount [80] provides (ϵ, δ)-differentially private statistics collection for the Tor network. It consists of three components: a Tally Server (TS), at least one Data Collector (DC), and at least one Share Keeper (SK). The TS, DCs, and SKs act collectively to report (noisy) counts of the events that were observed during a collection period. PrivCount supports both single number queries (*e.g.*, "how many clients connected during the collection period?") and multiple-counter queries (*e.g.*, "how many Tor connections went to {Google, Amazon, Facebook} during the collection period?") in which the "bins" (*i.e.*, counters) are independent. PrivCount is shown to provide (ϵ, δ)-differential privacy if at least one SK is honest.

We analyze the client statistics measured using the enhanced version of PrivCount (described in more detail by Mani et al. [102]) to better understand who uses Tor.

7.2 Enhancement to Private Set-union Cardinality (PSC)

We modify the implementation of PSC to include a Tally Server (TS). The TS is untrusted and merely coordinates the execution of the protocol: at the beginning of every measurement period the TS signals the CPs and DPs to start execution, and at the end all CPs and DPs send a *finished* signal to the TS. A malicious TS can collude with a malicious CP or DP and disrupt the flow of the protocol. However PSC's security guarantees ensure that, it cannot learn any information from an honest DP as long as at least one CP remains honest.

We also modify PSC to accept control-port events from a patched version of Tor (described by Mani et al. [102]), which allows PSC to securely count the number of distinct events observed across a set of relays. The DPs reside alongside the Tor software on Tor relays and perform event monitoring. During the collection period, each DP is notified of events from the patched version of Tor and maintains aggregated counts of these events in their "oblivious" counters. After the collection period, the DPs transmit their counter values via authenticated channels (over TLS) to the CPs. The CPs then collaboratively compute the (ϵ, δ)-differentially private cardinality of the union of the DPs' itemsets.

7.3 Methodology

Conducting safe measurements on Tor requires careful experimentation design, configuration, and deployment. In what follows, we describe the methodology we employ to collect and analyze measurements of the Tor network.

7.3.1 Deployment

We use several Tor relays and PSC to conduct Tor measurements. We use the patched version of Tor (described by Mani et al. [102]) to report additional information about connections, circuits, streams, and onion service directory usage information as necessary to answer our research questions. We ran 16 Tor relays with the patched version of Tor (six exit relays and ten non-exit relays). The Tor relays were run across three different countries – United States, Canada, and France – by three operators.

As discussed in §7.2, we slightly modify PSC to include a TS to coordinate the actions of the DCs and CPs. We engineer PSC to collect events emitted by our relays, and add support for measuring the number of unique domains (§7.4), clients (§7.5), and onion sites (§7.6). All of our enhancements have been merged into the PSC open-source project [100].

To better understand who uses Tor, we also analyze results – about client connections, circuits, and amount of data transferred – measured using the enhanced version of PrivCount [102]. We also analyze the geopolitical distribution and network diversity of the Tor clients.

We set up a PSC deployment containing one TS, three CPs, and 16 DPs (one for each Tor relay). We use PSC to repeatedly measure Tor in 24 hour periods, where each period focuses on measuring a small set of statistics. During some measurement periods, we use only a subset of the DPs and relays that are in a position to observe the events of interest in order to reduce operator overhead and reduce the likelihood of failure. Apart from the cases where servers were temporarily unavailable, the number of CPs we use is greater than or equal to the number of relay operators in all our measurements. To maintain our privacy guarantees, PSC measurements are never

Table 7.1: Action bounds for measurements.

Action	Daily bound	Defining activity
Connect to domain	20 domains	Web
Connect to Tor from new IP address	1 day: 4 IPs 2+ days: 2 IPs	N/A
Create TCP connection to Tor	12 connections	N/A
Create circuit through entry guard	651 circuits	Chat
Send or receive entry data	407 MB	Web
Upload descriptor of new onion address	3 addresses	Onionsite
Fetch descriptor	30 fetches	Onionsite

conducted in parallel, and we always enforce at least 24 hours of delay between any sequential measurement of distinct statistics.

7.3.2 Privacy

Recall from §3.2, that we provide event-level differential privacy using the methodology developed by Jansen and Johnson [80]. This approach bounds the amount of network activity protected by differential privacy within a certain length of time. Fortunately, reasonable network activity by any individual Tor user for most actions constitutes a small fraction of the total network activity. Therefore, we can provide privacy while providing accurate network measurements.

Our publishing mechanism formally applies (ϵ, δ)-differential privacy to the space of inputs containing all possible network traces in a given *epoch* (*i.e.*, measurement period). The inputs are considered to be "adjacent" if they differ in the

network activity of a single user within an epoch, and all differences in the activity stay within a set of *action bounds*. We use 24 hours as the length of an epoch, and our action bounds are shown in Table 7.1. One way to understand the privacy guarantee that results is that, for any two reasonable sequences of network actions that that a user could perform in an epoch, they most likely cannot be distinguished based on the statistics published by our mechanism.

We derive the action bounds by considering reasonable activities that a Tor user might perform over 24 hours. In this analysis, we consider protecting the privacy of both Tor clients and onion services. We take into account how certain type of user actions (*e.g.*, web browsing with Tor Browser, sending instant messages with the Ricochet P2P onion service [9], and running a Web server as an onionsite) translate into observable actions on the Tor network, such as creating new circuits or sending data cells, based on our understanding of the Tor protocol and software. We compute the amount of network actions that result from reasonable amounts of these activities and use the maximum to define the action bound. The activity that provides the maximum value defining each bound is shown in the final column of Table 7.1. The action bounds used during our Tor measurements are available for download [6].

For example, we choose to protect connecting to 20 unique domains through an exit circuit, which would protect browsing two new websites for each of 10 hours per day, as the other activities (*i.e.*, onionsites) would not create domain connections. This analysis also allows for additional page loads within the same site, as they would be assigned to the same circuit and thus would not be measured as a new domain connection. Note that certain observable actions apply to all Tor activities, such as creating a TCP connection to a guard, and thus have no defining activity.

We use privacy parameters $\epsilon = 0.3$ and $\delta = 10^{-11}$. The value of ϵ is the same one used by Tor to protect its onion service statistics [83]. We choose δ such that if there

are n Tor users, then δ/n is still small, which ensures that each user is simultaneously protected [59]. For example, if $n = 10^{-6}$ then $\delta/n = 10^{-5}$.

Note that the length of each measurement period only affects the accuracy of the results, as the differentially-private noise ensures privacy for any measurement length. We choose 24-hour periods to yield results that are accurate enough to draw conclusions from (*i.e.*, that are sufficiently large relative to the noise).

7.3.3 Statistical Analysis

Our measurements do not give us an exact and complete view of the total activity in the Tor network. This is because we perform measurements from a small subset of relays, and only observe a sample of Tor's overall activity. Moreover, PSC intentionally injects some random errors in the measurement. Therefore, we use statistical techniques to overcome these limitations.

The unique-count measurements made using PSC include noise generated according to a binomial distribution with known parameters. Aditionally hash-table collisions, that occur within PSC's internal data storage [67], can cause the measured value to be smaller than the true value. We adjust for these errors using an exact algorithm based on dynamic programming [102] that computes 95% confidence intervals.

Extrapolating unique counts from our sample to the entire network is challenging. Unlike standard counts, we need to know if the same items in our sample are observed elsewhere in the network or not. For example, when we count unique domains, it could be the case that the domains we count are each visited very often and thus is the set of domains seen by all the relays, and this would indicate that our sample count is the same as the network-wide count. In some cases, we can handle this using information about the frequency distribution of the observed items.

For example, there is evidence that domains visited follow a power-law distribution [24, 91]. We then make additional measurements to help determine likely parameters for these distributions. We construct confidence intervals for the network-wide unique counts by considering the probability of our local counts given different possible values for the overall count. We use Monte-Carlo simulations [102] to determine these probabilities for more complicated distributions.

Note that in some cases we cannot identify a likely frequency distribution. In these cases, given an observed value of x and a fraction p of all observations, we simply present the range of likely network-wide values to be $[x, x/p]$, where the lower end of the range covers the possibility that each item is observed frequently and the upper end covers the possibility of infrequent observations per item.

7.4 Exit Measurements

In this section, we present results and analyses of exit measurements taken from relays in a position to observe Tor's egress traffic.

7.4.1 Overview

Recall from §7.1.1 that Tor clients build *circuits* through a sequence of Tor relays over which they multiplex multiple *streams*. Each stream is associated with a TCP connection between an exit relay and a user-specified destination and is used to transfer data between that TCP connection and the client. For each stream, the client specifies a destination hostname or IP address which the exit relay requires in order to create a TCP connection on the client's behalf.

To better understand which web domains Tor users frequently visit, we focus on the *initial* streams that are created on circuits (*i.e.*, the first stream for each circuit)

Table 7.2: Locally observed unique second level domain statistics.

Statistic	Count	95% CI
SLDs	471,228	[470,357; 472,099]
Alexa SLDs	35,660	[34,789; 37,393]

Measured using PSC.

given that a *hostname* was provided in the stream by the client and the destination port requested by the client is a *web* port (either 80 or 443). We focus on initial streams because the Tor Browser uses a new circuit for each unique domain shown in the browser address bar. The initial stream on a circuit will therefore most directly indicate the user's intended destination, whereas subsequent streams that are created when loading a page to fetch embedded resources (*e.g.*, images, scripts, *etc.*) provide less useful indications of user intent. Second, we focus on streams that provide hostnames rather than IPv4 or IPv6 addresses because hostnames can be mapped to web domains much more easily than IP addresses. Finally, we are interested in web domains and so we do not measure domains on streams that request connections to destination ports not traditionally associated with web content.

7.4.2 Second Level Domains

We now describe the results from our measurements of the number of second level domains (SLDs) observed in initial streams that also provide a hostname and a web port. For our measurements, we increment the counter in our relays whenever we observe a second level domain that matches it. We conducted two related PSC SLD measurements. During both of these measurements, we use only five out of our six

exit relays (1.24% mean exit weight) in order to reduce operator overhead. We summarize our results in Table 7.2. In the first measurement, we measured the number of unique SLD names among those Tor primary domains which contain a TLD in the public suffix list [8]. From a measurement conducted between 2018-03-31 and 2018-04-01, we find that 471,228 (CI: [470,357; 472,099]) unique SLDs were accessed through our exits. We also measured the number of unique Alexa top 1 million SLDs (*i.e.*, the SLDs of Alexa top 1 million sites) among those Tor primary domains. From a measurement conducted between 2018-03-24 and 2018-03-25, we infer that 35,660 (CI: [34,789; 37,393]) of such unique SLDs were accessed through our exits. From these results, we find that the unique count of accessed SLDs is more than ten times that of the unique count of Alexa top one million sites. From our measurements, we conclude that a long tail exists in the distribution of sites accessed over Tor.

To extrapolate the unique second-level domain measurements *for the entire Tor network*, we need to know the distribution of SLDs as seen by the exits. From previous research studies, we know that domains are visited following a power-law distribution [24, 91]. But we need additional measurements to determine the exponent of this power-law distribution. Therefore, we perform a series of simulations of clients visiting random destination sites (based on power-law distribution with random exponents) and construct confidence intervals for the network-wide unique SLD counts. We use the locally observed unique SLDs count as a self-check.

This method appears to work well for the unique Alexa top 1 million SLDs. The results of 100 simulations reveals an inferred network-wide unique count of 513,342 (CI: [512,760; 514,693]) accesses to the Alexa top 1 million list. That is, slightly more than half of the Alexa top sites are accessed over Tor over 24 hours.

Table 7.3: Network-wide client usage statistics.

Statistic	Count	95% CI
Data (TiB)	517	± [504; 530]
Connections ($\times 10^6$)	148	± [143; 153]
Circuits ($\times 10^6$)	1,286	± [1,246; 1,326]

Inferred from PrivCount measurements.

Unfortunately, the inability to closely fit SLD accesses to a distribution prevented us from using this approach to extrapolate the number of network-wide SLD accesses.

7.5 Client Measurements

We conducted a number of measurements to better understand *who* uses Tor and to determine *how many* users access the network.

7.5.1 Tor Connections and Clients

We first analyze the client usage statistics measured using the enhanced version of PrivCount (refer §7.1.2). The network-wide inferred counts (with 1.4% mean entry weight) are summarized in Table 7.3.

The amount of daily data being transferred across Tor is 517 TiB (CI: [504; 530] TiB), which represents the sum of client uploads and downloads. We note that this includes Tor cell overheads, so the actual amount of client payload data transferred would be 2-3% less.

We find that there are about 148 million (CI: [143; 153] million) client connections, through which 1,286 million client circuits (CI: [1,246; 1,326] million) are multiplexed, per day, across the Tor network. This is significantly higher than

Table 7.4: Locally observed unique client statistics.

Statistic	Count	95% CI
IPs	313,213	[313,039; 376,343]
Countries	203	[141; 250]
ASes	11,882	[11,708; 12,053]
IPs (1-day)	306,905	[306,731; 343,832]
IPs (4-day)	670,955	[670,526; 890,786]
Churn (days 1-4)	121,350/day	[121,265; 182,318]/day
IPs (7-day)	885,778	[885,088; 1,341,759]
Churn (days 4-7)	71,608/day	[71,521; 150,324]/day

Measured using PSC.

the 80.6 million client connections reported by Jansen and Johnson [80] two years ago. We suspect that the discrepancy is due to ongoing distributed denial-of-service (DDoS) attacks that began affecting the Tor network back in December 2017 [76].

Tor Clients. We next measure the number of *unique* clients accessing the Tor network using PSC. Unlike existing work that also attempts to quantify the number of Tor users [37, 80, 99, 105], we do not store (even temporarily) IP addresses since PSC uses oblivious counters. Similar to previous studies, we assume that there is a one-to-one mapping between client IPs and unique Tor clients, although this may be violated, for example, by mobile users with changing IP addresses or by clients behind Network Address Translation (NAT). In addition, we count bridges as clients, as their identities are private.

We use PSC to collect data from our relays that have a non-zero guard probability, for a 24 hour period beginning on 2018-04-14 during which our guards had a combined weight of 1.19%. Measurement results are reported in Table 7.4.

We observe over 313 thousand unique client IPs using our guards (this experiment used two CPs due to the temporary unavailability of one). This is a surprisingly high number, given that Tor Metrics [130] (using a very different and unvalidated estimation technique) reports 2.15 million clients per day in April 2018 *for the entire Tor network*. We would expect a typical client to connect to three guards (clients currently use one guard for data but two additional guards for directory updates [10]), and so we would expect to see closer to $0.0119 \times 3 \times 2.15 \times 10^6 = 76,755$ unique clients IPs. To the extent that each unique IP observed in a 24-hour period represents a new user, this suggests that Tor is underestimating its total number of users by a factor of 4, and the total number of daily users is closer to $313,213/0.0119/3 \approx 8,773,473$.

To better understand the network-wide number of unique client IPs in Tor, we perform additional PSC measurements using sets of relays of *different sizes*. We can then compare how the count grows with the measuring set to its predicted growth under different client models to identify the most likely model. In particular we would like to validate a model of how many guards each client contacts, which we expect to typically be three, but could be less or more for several reasons, including guard/exit conflicts, multiple clients behind a single NAT IP, Tor bridges serving many clients, and tor2web instances [13]. We made two 24-hour measurements, one starting on 2018-05-12 using DPs with 0.42% of the guard weight and the other starting on 2018-05-13 using a disjoint set of DPs with 0.88% of the guard weight. We counted 148,174 (CI: [148; 161] thousand) and 269,795 (CI: [269; 315] thousand) unique client IPs during these measurements, respectively. Observe that the latter number is significantly smaller than we would expect if each client

Table 7.5: Network-wide promiscuous clients and client IPs.

Guards per client	Promiscuous clients 95% CI	Network-wide client IPs 95% CI
3	[15,856; 21,522]	[10,851,783; 11,240,709]
4	[15,129; 21,056]	[8,195,072; 8,493,863]
5	[14,428; 20,451]	[6,605,713; 6,849,612]

contacted to only one guard, in which case we would predict $148,174 \times (0.88/0.42) \approx 310,460$. This indicates that indeed client IPs typically connect to multiple guards.

To identify the number of guards that typical client IPs connect to, we consider a range of guards per client and analyze the resulting consistency with our measurements. Let g denote the number of guards each client connects to in the model. Using simulation [102] to extrapolate our two measurements to two separate confidence intervals for the network-wide unique client IP counts, we discover that these CIs are *disjoint* unless g is in the range $[27, 34]$. This is a much higher value of g than should be the case for typical clients, and we believe that this indicates that this is a poor model of how clients connect to guards. It does imply, however, that there are some clients IPs that are connected to many more guards than is typical.

Thus we consider a refinement on this model in which a set of "promiscuous" clients connects to *all* guards in a 24- hour period, and the remaining "selective" clients connect to g guards only. The promiscuous clients capture the likely behavior of Tor bridges, tor2web clients, and clients behind a NAT IP. We then determine a range for the number p of promiscuous clients by considering values of g that are likely given our understanding of the Tor protocol: $g \in \{3, 4, 5\}$ (all clients should connect to 3 guards for directory updates, some clients may connect to 1 or 2 more due to guard churn). For each of these values of g, using simulation [102], there

were some values for the number p of promiscuous clients that was consistent with our two client-IP measurements, that is, that yielded a non-empty intersection of the two CIs. Table 7.5 shows these ranges, which indicates about 14–22 thousand promiscuous clients.

Observe that bridges alone seem unlikely to make up this population, as this range is much larger than the approximately $1,640$ bridges reported by Tor Metrics at the time of our measurements, and in April 2016 Matic et al. [104] only discovered 50% more bridges than reported by Tor Metrics. Table 7.5 also shows confidence intervals for the number of network-wide client IPs, calculated for each g as the union of CIs over all p in the range shown. For what we believe is the most likely number of guards per selective client ($g = 3$), our measurements indicate about 11 million unique client IPs network-wide, which suggests a true value over 5 times the number of users estimated by Tor Metrics. Even for $g = 5$, the range suggests over 3 times as many daily users as Tor currently estimates.

Client Churn. Tor clients may go offline, and we expect that the Tor network experiences a high degree of client *churn* (*i.e.*, change in the connected client population). Using PSC, we recorded the count of unique client IPs over consecutive one-day, four-day, and seven-day period measurements beginning on 2018-07-02. We observed 306,905 (CI: [306,731; 343,832]), 670,955 (CI: [670,526; 890,786]), and 885,778 (CI: [885,088; 1,341,759]) unique client IPs, respectively. Comparing these results (see Table 7.4), we can conclude that the client churn rate is not constant. The client churn rate is on average 121,350 (CI: [121,265; 182,318]) client IPs per day for the first three days, and 71,608 (CI: [71,521; 150,324]) client IPs per day for the next three days.

We observe that the client churn rate decreases. We can consider a few possible explanations for this. The first is that some decreasing number of "new" clients

connects to our guards everyday. This seems unlikely, as even if the rate of new users joining Tor has some periodic behavior, we observed a decreasing churn rate with two consecutive measurements of different lengths. A more likely explanation is that some clients (*e.g.*, mobile users) have a small pool of IP addresses from which they connect to Tor. Over time, their guards observe an increasing fraction of that pool, and the chance that an unseen IP address is used decreases. Another possible explanation is that there is some number of promiscuous clients, but they take longer than a day to connect to each guard at least once.

7.5.2 Client Composition and Diversity

We next explore the geopolitical distribution of Tor clients. We analyze the number of client connections, bytes transferred, and circuits created broken down by country, measured using the enhanced version of PrivCount (refer §7.1.2) by resolving each client IP address to its host country using the MaxMind GeoLite2 country database. For most of the world's 250 countries, the added noise (to achieve differential privacy) overwhelms the actual count. Therefore, we only show the countries with significant inferred counts (with 1.4% mean entry weight) in Figure 7.2.

The United States (US), Russia (RU), and Germany (DE) have both the greatest client connection counts and data transfer amounts (see Figure 7.2, *left* and *center*). Assuming that the distribution of client connection counts reflects the distribution of client usage, we can surmise that these three countries use Tor the most.

Interestingly, our results disagree with those from the Tor Metrics Portal, which ranks the United Arab Emirates (UAE) as contributing the second largest number of Tor users; in contrast, we did not find the UAE to be among the most significant contributors. Although we cannot fully explain this discrepancy, we are somewhat skeptical of the Tor Metric Portal's results: the value reported by the Tor Metrics

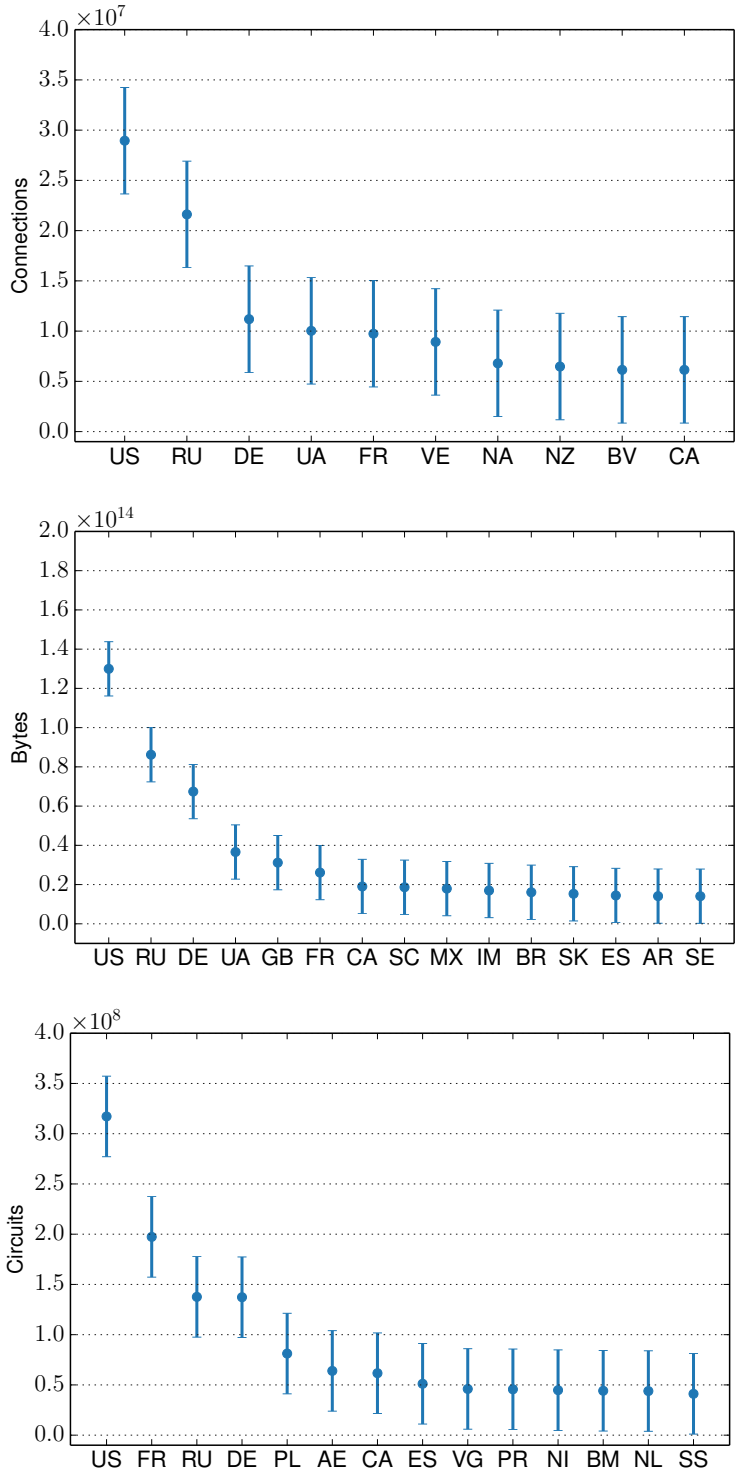

Figure 7.2: Network-wide per country client usage statistics. Tor per country client usage statistics inferred from PrivCount measurements - the counts of client connections (*top*), amount of client data transferred (*middle*), and the counts of client circuits (*bottom*). Error bars show 95% confidence intervals.

Portal implies that nearly 4% of the entire population of the UAE accesses Tor on a daily basis, which seems unlikely.

And, surprisingly we also observe significant client connection count, client data transferred, or number of client circuits from many scarcely populated/uninhabited islands, such as, Bouvet island (BV), Seychelles (SC), The Isle of Man (IM), The British Virgin Islands (VG), and Bermuda (BM). Though we are unable to explain this discrepancy, one intriguing possibility is that these connections are from Tor bridges, single onion services, or Tor network scanners.

To determine whether Tor has a global userbase, we use PSC to count the number of unique countries from which clients originate. Since the actual count can only be at most 250 (*i.e.*, the total number of countries worldwide), invariably the differentially private noise overwhelms the actual count. Therefore to reduce the effects of noise, we average the country counts between two consecutive one-day measurements, beginning on 2018-05-09. We find that clients from 203 (CI: [141; 250]) different countries access Tor. Comparing our results to earlier studies from McCoy et al. [105] (2008) and Chaabane et al. [37] (from 2010) that report clients from approximately 125 countries, we can conclude that the locations from which Tor is accessed has significantly diversified in the past decade.

We were unable to extrapolate the unique country count for the entire Tor network since the frequency distribution of the countries, as seen by the guards, is difficult to determine. However, we can still roughly estimate the range of network-wide values to be [141, 250], where the lower bound is the least possible observed count and the upper bound is the total number of countries that exists worldwide.

Network Diversity. We next explore the network diversity of Tor clients by mapping each client IP address to its autonomous system (AS) using the IPv4 and IPv6 datasets (dated 26th November 2017) from CAIDA [1], consisting of 59,597

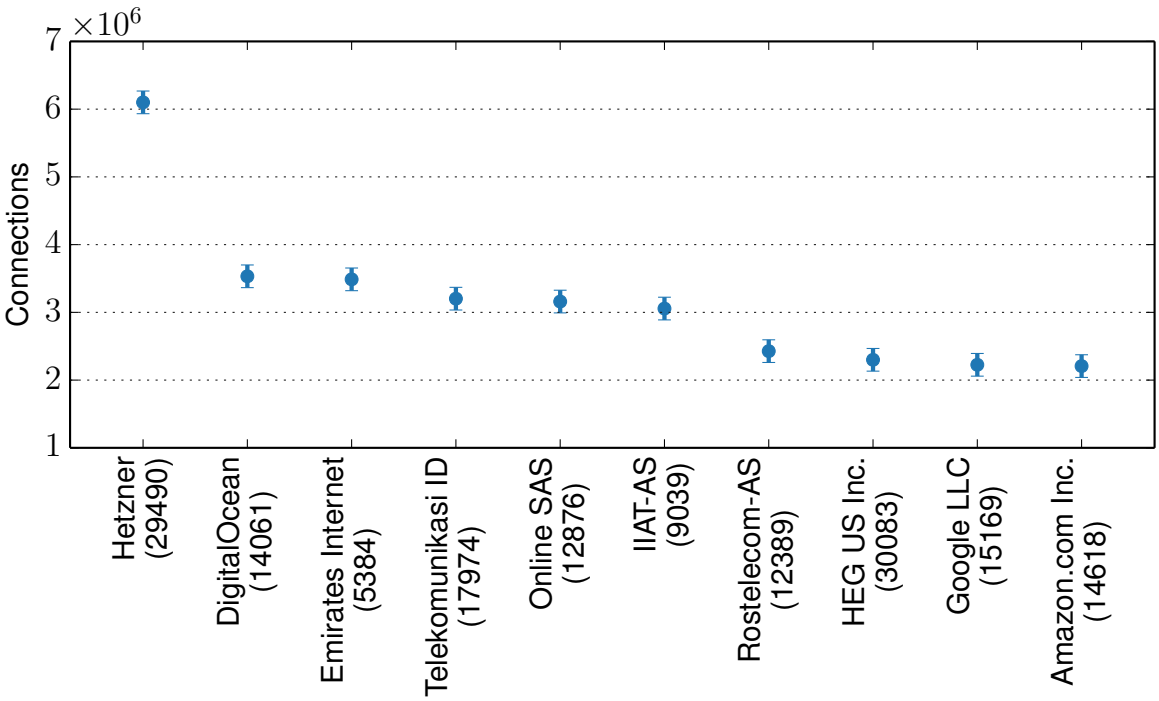

Figure 7.3: Network-wide client connection count for Top 10 ASes. Top 10 ASes' client connection count inferred from PrivCount. Error bars with 95% confidence intervals are very close to the plotted value, and are not visually discernable.

ASes. Using PSC, we first measure the unique AS count of the client IPs, for a 24 hour period beginning on 2018-04-18. We observe clients from about 11,882 ASes (CI: [11,708; 12,053])—approximately 20% of the number of defined ASes.

To extrapolate the unique AS count *for the entire Tor network*, we must first identify the frequency distribution of the ASes, as seen by the guards. This is challenging. Therefore, we present the range of most-likely network-wide values as [11,708; 59,597], where the lower bound is the least possible observed count and the upper bound is the total number of ASes.

To determine whether there are any "hotspot" ASes, we analyze the client connection count for each AS measured using the enhanced version of PrivCount (see §7.1.2). Since the total number of ASes (59,597) is large, the per-AS differentially-

Table 7.6: Locally observed unique v2 onion address statistics.

Statistic	Count	95% CI
Addresses published	3,900	[3,769; 4,045]
Addresses fetched	2,401	[1,101; 3,718]

Measured using PSC.

private noise is by chance large for some of them. Therefore to reduce the effect of noise, the measurement is run in two phases (as described in [102]), where the first phase acts as a filter to reduce the ASes under consideration to those with likely positive true counts for the second phase. The network-wide inferences (with 1.2% mean entry weight) of the resulting top 10 ASes is plotted in Figure 7.3. We observe that, almost 22% of Tor's entry connections are confined to these 10 ASes. A surprising number of these ASes are hosting providers (*e.g.*, Hetzner, DigitalOcean, Online SAS), from which we would not expect consumer connections. We suspect that these entry connections are from a mix of Tor bridges, onion services, Tor2web proxies, and Tor network scanners. Also, the unexplainable discrepancies in Figure 7.2, could be a reflection of the high number of hosting provider client connections.

7.6 Onion Service Measurements

Recall from §7.1.1 that Tor onion services store their descriptors at onion service directories (HSDirs) in a DHT. To measure the number of *unique* onion services in the Tor network, we instrument our HSDirs to report the onion service *address* for each version 2 (v2) [11] descriptor that is *published* to and *fetched* from the DHT. (We don't measure version 3 (v3) onion service descriptors because the onion address is

obscured using key blinding [12].) We use our PSC deployment to safely capture the approximate number of v2 onion services observed by our HSDirs. Unlike existing work that also attempts to quantify the number of onion services [77, 82, 83, 114], we avoid the need to store (even temporarily) onion addresses, since PSC uses oblivious counters. Table 7.6 summarizes our onion service address measurements.

Unique Onion Service Addresses Published. From a PSC measurement conducted between 2018-04-23 and 2018-04-24 during which our mean combined HSDir publish weight was 2.75%, we observed 3,900 (CI: [3,769; 4,045]) unique v2 onion addresses in descriptors published to our HSDirs. We extrapolate these results based on HSDir replication [102] to infer that the total number of unique onion service addresses published *for the entire Tor network* is 70,826 (CI: [65,738; 76,350]). Accordingly, our relays observed at least 4.93% of Tor's unique onion addresses. Using a different extrapolation method [77], the Tor Metrics Portal [130] estimates a total of 79 thousand unique v2 onion services published in the Tor network at the time of our measurement. The Tor metrics estimate does not include a confidence interval which we expect to be significant since *every* reporting relay adds noise (due to the lack of secure aggregation of Tor metrics reports). Therefore, we expect that our confidence intervals would overlap.

Unique Onion Service Addresses Fetched. From a PSC measurement conducted between 2018-04-29 and 2018-04-30 during which our mean combined HSDir fetch weight was 0.534%, we observed 2,401 (CI: [1,101; 3,718]) unique v2 onion addresses in descriptors fetched from our HSDirs. We extrapolate these results based on HSDir replication [102] to infer that the total number of unique onion service addresses fetched *for the entire Tor network* is 74,900 (CI: [34,363; 696,255]). By comparing the number of unique onion addresses published and fetched, we estimate that between 45% and 100% of active onion services are used by clients. Note

that the Tor Metrics Portal [130] does not estimate the number of unique onion addresses fetched by clients, and its estimate of the number of unique v2 onion addresses published occasionally varies significantly (*e.g.*, it increased by 40 thousand during our measurement).

7.7 Ethical and Safety Considerations

Statistics collection in the setting of an anonymity network inherently carries risk since the exposure of information could degrade the privacy of the network's users and potentially subject them to harm. Guiding our study were the four criteria for ethical network research established by the Menlo Report [54]: we uphold the principle of *respect for persons* through the careful and principled application of data protections, including the use of differentially private techniques and the avoidance of collecting personally identifiable information (*e.g.*, IP addresses). Following the principle of *beneficence*, we balance the (low) risk of harm with the potential benefits of the research – namely, an increased understanding of the Tor network that could inform research on improving the network's security and performance. Our techniques achieve the Menlo Report's notion of *justice*, since our statistics do not target a specific subpopulation of Tor's users. Finally, we achieve *respect for law and public interest* through transparency in our methods: we use an open source tool [67] for statistics collection and we submitted our research plan (prior to its implementation) to several review bodies. We presented it to the Tor Research Safety Board [132], which concluded that our plan "provides a plausible strategy for safely measuring trends on the Tor network". The full text of the TRSB's findings is available online [6]. Our measurement study was approved by the ethics board at the University of New South Wales and certified as being exempt as non-human subjects research

by the Institutional Review Board (IRB) at Georgetown University. Additionally, aspects of this study were discussed with members of Georgetown's Office of General Counsel, who raised no concerns.

7.8 Discussion and Limitations

A challenge – and a limitation – of our measurement approach is the difficulty of "getting above the noise." Our differentially private measurement techniques inject a quantity of noise that is at least sufficient to ensure that all differences in monitored activities fall within a set of action bounds. Sometimes, the level of noise is such that the signal-to-noise ratio (SNR) diminishes or even eliminates our ability to draw meaningful conclusions. For example, when examining the number of unique countries from which Tor clients originate, invariably the differentially private noise overwhelms the actual count; we find that even after averaging over two one-day measurements, our confidence interval (CI: [141; 250]) is significantly large. While future (and more optimized) privacy-preserving measurement techniques may offer greater SNRs, we posit that certain counts, such as, unique country count may inherently be too small to usefully measure without sacrificing privacy.

Our study does not attempt to distinguish between human-driven and automated activity on the Tor network. An intriguing open problem is the construction of techniques to identify when measurements are heavily influenced by automated activities (for example, when swarms of infected hosts belonging to a botnet connect to Tor). Indeed, the enormously high connections from hosting providers and uninhabited/scarcely populated islands suggest that automated behavior likely does occur on Tor. Our hope is that measurement studies such as this will help inform developers and security researchers of unexpected automated activities – a necessary pre-

requisite to understanding and potentially defending against malicious automated processes. Moreover, our findings can also aid in developing better Tor network traffic and user behavior models that can be used to improve Tor's performance and security.

7.9 Conclusion

This chapter presents a comprehensive privacy-preserving measurement study of the live Tor network. We observe a surprisingly large number of unique client IP addresses, suggesting that the heuristically-derived estimates from the Tor Metrics Portal are significantly underreporting Tor's userbase. We find that the network's clients are highly distributed, connecting from more than 200 countries and nearly 12,000 ASes. These clients construct more than 1.2 billion anonymous circuits per day, carrying approximately 517 TiB of data (6.1GiB/s). Some of the data (for our findings) is available for download [6].

Chapter 8

Conclusion

This thesis first explores the simple problem of evaluating the *behavior* of anonymity networks. It presents an extensive evaluation of the Internet's open proxies, that are often used as an alternative for achieving a weak form of sender anonymity. We find that open proxies do achieve a limited degree of anonymity. However, the lack of any security guarantees for traffic passing through these proxies makes their use highly risky: users may unintentionally expose their traffic to malicious manipulations, especially when no end-to-end security mechanisms (*e.g.*, TLS) are present. Our experiments disclose and quantify new forms of misbehavior, reinforcing the notion that open proxies should be used with extreme caution.

As a point of comparison, this thesis also presents a similar monthlong experimental study of the Tor network and compares the level of manipulation observed using open proxies to that when the content is fetched over Tor. We found zero instances of misbehavior and far greater stability and goodput, indicating that Tor offers a safer and more reliable form of anonymous communication.

This thesis next explores the much more complex problem of measuring the *usage* of anonymity networks. It presents efficient, safe and distributed (ϵ, δ)-differentially private systems – HisTorϵ and PSC– for gathering user statistics in

anonymity networks (primarily Tor). Like existing schemes [62, 80], HisTorϵ and PSC also provide resistance to "compulsion attacks", wherein a relay operator can be forced to reveal logs through a subpoena. Unlike existing work, PSC efficiently aggregates the count of unique items recorded across a set of relays privately and HisTorϵ provides strong integrity guarantees. We demonstrate that both HisTorϵ and PSC operate with low computational overhead and reasonable bandwidth. Though these systems are primarily designed for Tor, they also have wide applicability and can be in principle used to collect statistics privately from any other relay based anonymity networks as well.

Finally, we deploy PSC on the live Tor network and perform an in-depth measurement study of Tor, with user privacy as the foremost consideration. Our findings both reinforce existing beliefs about the Tor network (*e.g.*, the number of unique onion addresses published) and elucidate many aspects of the Tor network that have previously not been explored. Perhaps the most interesting finding is that we observe a surprisingly large number of unique client IP addresses, suggesting that the heuristically-derived estimates from the Tor Metrics Portal are significantly underreporting Tor's userbase. We hope that our measurements can aid in developing better network traffic and user behavior models that can be used to improve Tor's performance and security.

References

[1] Routeviews prefix-to-as mappings (pfx2as) for ipv4 and ipv6. http://data.caida.org/datasets/routing/routeviews-prefix2as/.

[2] kyber: Dedis advanced crypto library for go. https://github.com/dedis/crypto.

[3] The invisible internet project. https://geti2p.net/en/.

[4] Jondo inc. jondonym. http://anonymous-proxy-servers.net/.

[5] libnum library for gm encryption. https://github.com/hellman/libnum.

[6] Privacy-preserving tor network measurements. https://onioncount.github.io/.

[7] Go par package for parallel for-loops. https://github.com/danieldk/par.

[8] Public suffix list. https://publicsuffix.org.

[9] Ricochet. https://ricochet.im/.

[10] Tor directory protocol, version 3. https://gitweb.torproject.org/torspec.git/plain/dir-spec.txt, .

[11] Tor rendezvous specification (version 2). https://gitweb.torproject.org/torspec.git/plain/rend-spec-v2.txt, .

[12] Tor rendezvous specification (version 3). https://gitweb.torproject.org/torspec.git/plain/rend-spec-v3.txt, .

[13] Tor2web: Browse the tor onion services. https://www.tor2web.org, .

[14] Akamai's state of the internet q2 2015 report, 2015. Available at https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/2015-q2-cloud-security-report.pdf.

[15] Proxy list, 2018. Available at https://raw.githubusercontent.com/clarketm/proxy-list/master/proxy-list.txt.

[16] Coinhive: A crypto miner for your website, 2018. Available at https://coinhive.com/.

[17] Monero: Private digital currency, 2018. Available at https://getmonero.org/.

[18] Free anonymous proxy list – all proxies, 2018. Available at http://multiproxy.org/txt_all/proxy.txt.

[19] Free anonymous proxy list – anonymous proxies, 2018. Available at http://multiproxy.org/txt_anon/proxy.txt.

[20] Nordvpn, 2018. Available at https://nordvpn.com/.

[21] Ssl blacklist (sslbl) - a list of "bad" ssl certificates, 2018. https://sslbl.abuse.ch/.

[22] Workingproxies.org. the best free proxy server list on the web, 2018. Available at `http://www.workingproxies.org/plain-text`.

[23] Xroxy: More than just proxy, 2018. Available at `http://www.xroxy.com/proxylist.htm`.

[24] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3(1), 2002.

[25] D. Akhawe and A. P. Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX security symposium*, 2013.

[26] G. Asharov and Y. Lindell. A full proof of the bgw protocol for perfectly-secure multiparty computation. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, 2011.

[27] S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012.

[28] O. Berthold, H. Federrath, and M. Köhntopp. Project "anonymity and unobservability in the internet". In *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*. ACM, 2000.

[29] A. Bhattachayya. On a measure of divergence between two statistical population defined by their population distributions. *Bulletin Calcutta Mathematical Society*, 35, 1943.

[30] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. 2013.

[31] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann. Content and popularity analysis of tor hidden services. In *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on*. IEEE, 2014.

[32] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In *Symposium on Theory of Computing*, STOC '08, 2008.

[33] F. Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In *International Conference on Information Security and Cryptology*. Springer, 2005.

[34] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1(101101), 2010.

[35] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 2001.

[36] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Symposium on Theory of Computing (STOC)*, 2002.

[37] A. Chaabane, P. Manils, and M. Kaafar. Digging into anonymous traffic: A deep analysis of the tor anonymizing network. In *IEEE Network and System Security*, 2010.

[38] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis. Detecting traffic snooping in tor using decoys. In *Recent Advances in Intrusion Detection (RAID)*, 2011.

[39] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology (CRYPTO '92)*, 1993.

[40] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[41] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[42] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards statistical queries over distributed private user data. In *NSDI*, volume 12, 2012.

[43] R. Chen, I. E. Akkus, and P. Francis. Splitx: high-performance private analytics. In *ACM SIGCOMM Computer Communication Review*, volume 43. ACM, 2013.

[44] Constverum. Proxybroker, 2018. Available at `https://github.com/constverum/ProxyBroker`.

[45] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017.

[46] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015.

[47] T. M. Dalton. Affidavit of special agent thomas m. dalton. Available at `https://cbsboston.files.wordpress.com/2013/12/kimeldoharvard.pdf`.

[48] I. Damgard, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology (CRYPTO 2012)*, 2012.

[49] G. Danezis and C. Diaz. A survey of anonymous communication channels. 2008.

[50] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security (CANS 2012)*, 2012.

[51] R. Dingledine and N. Mathewson. Tor protocol specification. `https://spec.torproject.org/tor-spec`.

[52] R. Dingledine and S. Murdoch. Performance improvements on tor, or, why tor is slow and what we're going to do about it. `https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf`, March 2009.

[53] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[54] D. Dittrich, E. Kenneally, et al. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, US Department of Homeland Security, August 2012.

[55] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4), 1983.

[56] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson. The security impact of https interception. In *Network and Distributed Systems Symposium (NDSS)*, 2017.

[57] C. Dwork. Differential privacy. *Automata, Languages and Programming*, 2006.

[58] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference (TCC)*, 2006.

[59] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4), 2014.

[60] M. Edman and B. Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys (CSUR)*, 42 (1), 2009.

[61] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *Australasian Conference on Information Security and Privacy*. Springer, 2015.

[62] T. Elahi, G. Danezis, and I. Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[63] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4), 1985.

[64] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014.

[65] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring https adoption on the web. In *26th USENIX Security Symposium*, August 2017.

[66] E. Fenske, A. Mani, A. Johnson, and M. Sherr. Private set-union cardinality with full accountability. `https://security.cs.georgetown.edu/projects/`.

[67] E. Fenske, A. Mani, A. Johnson, and M. Sherr. Distributed measurement with private set-union cardinality. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017.

[68] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology CRYPTO '86*. Springer, 1986.

[69] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. May 2004.

[70] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *International Conference on Financial Cryptography*. Springer, 2002.

[71] E. Gabber, P. B. Gibbons, Y. Matias, and A. Mayer. How to make personalized web browsing simple, secure, and anonymous. In *International Conference on Financial Cryptography*. Springer, 1997.

[72] N. Gelernter, A. Herzberg, and H. Leibowitz. Two cents for strong anonymity: The anonymous post-office protocol. *Proceedings on Privacy Enhancing Technologies*, 2, 2016.

[73] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge university press, 2009.

[74] S. Goldwasser and Y. Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3), 2005.

[75] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2), 1984.

[76] D. Goulet. tor-project listserv post, available at `https://lists.torproject.org/pipermail/tor-project/2017-December/001604.html`, 2017.

[77] D. Goulet, A. Johnson, G. Kadianakis, and K. Loesing. Hidden-service statistics reported by relays. Technical Report 2015-04-001, The Tor Project, Inc., April 2015.

[78] J. Groth. A verifiable secret shuffe of homomorphic encryptions. In *International Workshop on Public Key Cryptography*. Springer, 2003.

[79] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. *Journal of Cryptology*, 25(3), July 2012.

[80] R. Jansen and A. Johnson. Safely measuring tor. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[81] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. Inside job: Applying traffic analysis to measure tor from within. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[82] G. Kadianakis and K. Loesing. Extrapolating network totals from hidden-service statistics. Technical Report 2015-01-001, The Tor Project, January 2015.

[83] G. Kadianakis, D. Goulet, K. Loesing, and A. Johnson. Better hidden service stats from tor relays. `https://gitweb.torproject.org/torspec.git/tree/proposals/238-hs-relay-stats.txt`, 2014.

[84] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing (STOC)*, 1997.

[85] S. P. Kasiviswanathan and A. Smith. On the 'semantics' of differential privacy: A bayesian formulation. *Journal of Privacy and Confidentiality*, 6(1), 2014.

[86] D. Kedogan, D. Agrawal, and S. Penz. Limits of anonymity in open environments. In *International Workshop on Information Hiding*. Springer, 2002.

[87] R. Khare and S. Lawrence. Upgrading to tls within http/1.1. Request for Comments (RFC) 2817, Internet Engineering Task Force, May 2000.

[88] L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings on Advances in Cryptology (CRYPTO 2005)*, 2005.

[89] D. Koblas and M. R. Koblas. Socks. In *USENIX UNIX Security III Symposium*, 1992.

[90] S. Köpsell and U. Hillig. How to achieve blocking resistance for existing systems enabling anonymous web surfing. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, 2004.

[91] S. A. Krashakov, A. B. Teslyuk, and L. N. Shchur. On the universality of rank distributions of website popularity. *Computer Networks*, 50(11), 2006.

[92] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2), 2016.

[93] E. Larraia, E. Orsini, and N. Smart. Dishonest majority multi-party computation for binary circuits. In *Advances in Cryptology (CRYPTO 2014)*, 2014.

[94] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for voip systems. In *ACM SIGCOMM Computer Communication Review*, volume 45. ACM, 2015.

[95] J. Lee and C. Clifton. How much is enough? choosing ε for differential privacy. In *International Conference on Information Security*, 2011.

[96] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. Socks protocol version 5. Request for Comments (RFC) 1928, Internet Engineering Task Force, March 1996.

[97] LetsEncrypt. Percentage of web pages loaded by firefox using https, 2018. Available at `https://letsencrypt.org/stats/`. Retrieved 9 June, 2018.

[98] Y. Lindell, B. Pinkas, N. Smart, and A. Yanai. Efficient constant round multi-party computation combining bmr and spdz. In *Advances in Cryptology (CRYPTO 2015)*, 2015.

[99] K. Loesing, S. J. Murdoch, and R. Dingledine. A case study on measuring statistical data in the tor anonymity network. In *Financial Cryptograpy and Data Security*, 2010.

[100] A. Mani. Private set-union cardinality. https://github.com/AkshayaMani/PSC.

[101] A. Mani and M. Sherr. Historϵ: Differentially private and robust statistics collection for tor. In *Network and Distributed System Security Symposium (NDSS)*, February 2017.

[102] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. Understanding tor usage with privacy-preserving measurement. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 2018.

[103] N. Mathewson. Some thoughts on hidden services (tor blog post), 2014. Available at https://blog.torproject.org/blog/some-thoughts-hidden-services.

[104] S. Matic, C. Troncoso, and J. Caballero. Dissecting tor bridges: a security evaluation of their private and public infrastructures. In *Network and Distributed Systems Security Symposium*, 2017.

[105] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining light in dark places: Understanding the tor network. In *International Symposium on Privacy Enhancing Technologies Symposium*, 2008.

[106] F. McSherry and R. Mahajan. Differentially-private network trace analysis. *ACM SIGCOMM Computer Communication Review*, 41(4), 2011.

[107] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Foundations of Computer Science (FOCS'07)*, 2007.

[108] L. Melis, G. Danezis, and E. De Cristofaro. Efficient private statistics with succinct sketches. *arXiv preprint arXiv:1508.06110*, 2015.

[109] D. Moore and T. Rid. Cryptopolitik and the darknet. *Survival*, 58(1), 2016.

[110] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, November 2001.

[111] L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. In *Applied Cryptography and Network Security (ACNS)*. Springer, 2004.

[112] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Symposium on Theory of Computing*, STOC '07, 2007.

[113] NIST. Recommended elliptic curves for federal government use, 1999. Available at `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`.

[114] G. Owen and N. Savage. Empirical analysis of tor hidden services. *IET Information Security*, 10(3), 2016.

[115] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the web: An open proxy's view. *SIGCOMM Comput. Commun. Rev.*, 34(1), Jan. 2004.

[116] M. Perry. The trouble with cloudflare (tor blog post), March 2016. Available at `https://blog.torproject.org/blog/trouble-cloudflare`.

[117] A. Pfitzmann and M. Waidner. Networks without user observability - design options. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1985.

[118] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis. The loopix anonymity system. In *26th USENIX Security Symposium, USENIX Security*, 2017.

[119] M. Prince. The trouble with tor (cloudflare blog post), March 2016. Available at `https://blog.cloudflare.com/the-trouble-with-tor/`.

[120] J. R. Rao and P. Rohatgi. Can pseudonymity really guarantee privacy? In *USENIX Security Symposium*, 2000.

[121] J. Ren and J. Wu. Survey on anonymous communications in computer networks. *Computer Communications*, 33(4), 2010.

[122] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3), 1991.

[123] W. Scott, R. Bhoraskar, and A. Krishnamurthy. Understanding open proxies in the wild. *Chaos Communication Camp*, 2015.

[124] C. Soghoian. Researchers could face legal risks for network snooping. `http://www.cnet.com/news/researchers-could-face-legal-risks-for-network-snooping/`, July 2008.

[125] C. Soghoian. Enforced community standards for research on users of the tor anonymity network. In *Workshop on Ethics in Computer Security Research (WECSR)*, 2011.

[126] K. Steding-Jessen, N. L. Vijaykumar, and A. Montes. Using low-interaction honeypots to study the abuse of open proxies to send spam. *INFOCOMP Journal of Computer Science*, 7(1), 2008.

[127] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1), 2003.

[128] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *USENIX security symposium*, 2009.

[129] Tor Project. Tor: Onion service protocol. `https://www.torproject.org/docs/onion-services.html.en`.

[130] I. Tor Project. Tor metrics portal, . `https://metrics.torproject.org/`.

[131] I. Tor Project. Who uses tor?, . `https://www.torproject.org/about/torusers.html`.

[132] Tor Research Safety Board. Available at `https://research.torproject.org/safetyboard.html`.

[133] G. Tsirantonakis, P. Ilia, S. Ioannidis, E. Athanasopoulos, and M. Polychronakis. A large-scale analysis of content modification by open http proxies. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[134] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017.

[135] G. Tyson, S. Huang, F. Cuadrado, I. Castro, V. C. Perta, A. Sathiaseelan, and S. Uhlig. Exploring http header manipulation in-the-wild. In *International Conference on World Wide Web (WWW)*, 2017.

[136] J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4), 2005.

[137] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015.

[138] K. Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*, 72, 2008.

[139] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled onions: Exposing malicious tor exit relays. In *Privacy Enhancing Technologies Symposium*, 2014.

[140] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, 2012.