Abstract

Design and Implementation of Privacy-Preserving Surveillance

Aaron Segal

Yale University

2016

The modern internet and phone networks offer very little security, privacy, or accountability to their users. As people conduct their business and social lives online and over the phone, they naturally generate private or sensitive data about themselves. But any number of parties can and do track this data. Not only the services people interact with everyday, but third-party services for ad tracking, malicious hackers, government agencies operating with nebulous legal authority, and service providers themselves can and do observe and track users. They can then use the sensitive data in a variety of objectionable ways.

Changing this state of affairs without an earth-shattering technological breakthrough may appear to be a hopeless situation. But, in this dissertation, we demonstrate how existing technology can, if deployed and used properly, markedly improve privacy for users and accountability for those collecting data. We discuss two techniques for achieving these improvements: privacy-preserving surveillance and anonymous communication. For each technique, we present example protocols for which we have have implemented fast prototypes running on commercial hardware.

First, we define the notion of privacy-preserving surveillance. Currently, a government agency can collect and examine bulk user data while making no distinction between the legitimate target of investigation and the average person, and with little or no oversight from other agencies. Privacy-preserving surveillance is an alternative

legal regime in which searches of sensitive user data could only take place with the active collaboration of multiple government agencies. Trust is distributed amongst these agencies, assuring that no single authority can unilaterally view sensitive user data (or metadata). We then show how two types of bulk surveillance, currently in use by the authorities, could be made privacy-preserving by the adoption of modern cryptographic protocols to secure data.

We also discuss protocols for anonymous communication. We take two approaches to anonymity. First, we present an improvement to the Tor network, an anonymity substrate based on onion routing that is already deployed in the wild. Second, we present a complete specification of the dining-cryptographers-based Verdict protocol and formally prove its anonymity, security, and accountability properties.

Design and Implementation of Privacy-Preserving Surveillance

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Aaron Segal

Dissertation Director: Joan Feigenbaum

December 2016

Contents

List of Figures

List of Tables

Without my father, I would never have learned to write programs.

Without my mother, I would never have learned to write English.

This thesis is dedicated to them.

Acknowledgments

I would sincerely like to thank my advisor, Prof. Joan Feigenbaum, for her brillant teaching, wise advice, and continual encouragement and support. I'm also deeply grateful for her generous support of graduate student social life as chair of the Department of Computer Science.

Thank you to Prof. Bryan Ford for his insightful feedback and assistance with my research in privacy-preserving surveillance.

Thanks to Dr. Aaron Johnson, Dr. Rob Jansen, and Dr. Paul Syverson of the US Naval Research Laboratory for hosting me over the summer at NRL, and for collaborating with me on our under-appreciated research into Tor.

I owe a debt of gratitude to my undergraduate research advisor, Prof. Leonid Reyzin of Boston University, for opening my eyes to the world of computer science reseach. Without him, I wouldn't even be in this field.

For keeping me going these last five years, I'd to thank the friends I made at Yale, including but not limited to: Henny Admoni, Will Dower, Debayan Gupta, Brad Hayes, Jérémie Koenig, Rasmus Kyng, Alex Litoiu, David Meierfrankenfeld, Lawrence Moy, Nathan Schwalm, Mark Schwab, Amelinda Webb, and Andrew Womack. And I'd like to thank my friends who aren't at Yale, but stood by me anyway: Dan and Anna Copel, Kristine Gammer, Siggy and Kate Tomascovic-Moore, Rachel Lucas, Bart Moore, Luke Moreau, Barry Torch, and my brother Jacob Segal.

The research in this dissertation was supported by a Computer Science Department Kempner Fellowship, a grant from the Office of Naval Research, and a Google Faculty Research Award.

Chapter 1

Introduction

As people conduct their business and social lives online and over the phone, they naturally generate private or sensitive data about themselves. This data includes content that users generate directly, such as email, social media posts, photos, and videos; metadata, such as phone records and lists of contacts; and tracking data, such as the websites a user visits and the cell towers a mobile phone connects to.

Unfortunately, the modern internet and phone networks offer people very little control over their data. Most people have very little expectation that their sensitive information will remain private. In the case of metadata and tracking data, many people do not even realize how much about them there is to know, or who knows it.

In fact, a number of parties can and do collect sensitive, personal data in bulk, and use this data in dangerous or even malicious ways. Advertisers online commonly use data about individuals to target ads. This practice can be relatively benign, but it can also allow shared users of a computer to discover facts about each other from the ads served to their shared browser, even if they clear their internet history [9]. Criminals can use sensitive data about their targets to commit fraud or blackmail, or to enhance the effectiveness of a phishing attack [30]. Government agencies also

collect sensitive data, often operating under nebulous, classified authorities, and use the data in poorly supervised investiations.

The situation may look hopeless. Given the government interest in observing and collecting private information, it may appear that the only way for average internet and phone users to have any kind of privacy would be the invention of some great new cryptographic wonder. Some hold out hope for the invention of a technology that would allow trustworthy government investigators, and only them, unfettered access to user data - although this solution would still have massive privacy implications without sufficient oversight. In the context of surveillance, some go so far as to say that user privacy *must* be abandoned in favor of national security.

But the truth is that the cryptographic tools to improve privacy, security, and accountability already exist.

In this thesis, we will discuss two techniques that could be used to improve users' control over their private information: privacy-preserving surveillance, and anonymity. Both rely only on well-known and well-studied systems of encryption and authentication, and could be deployed now, at scale and on commercial hardware. Tools for anonymous communication over the internet are starting to see use, primarily in the form of a program called Tor [19], but Tor has some significant issues we address in this thesis, both with improvements to the Tor network and with an alternative anonymity system which we prove provides anonymity, accountability, and integrity. Privacy-preserving surveillance, if adopted along with a legal framework supporting it, would allow investigatory agencies to conduct warranted searches of metadata without destroying the privacy of all internet or phone users in the process, and ensure oversight for such searches to prevent abuses.

1.1 Outline of this Thesis

In Chapter 2, we discuss the principles behind privacy-preserving surveillance. We will present several specific policies that should be used by governments engaging in electronic bulk surveillance activities to make the surveillance process accountable, limited in scope, and precisely targeted.

We will then discuss the technical basis for privacy-preserving surveillance in Chapters 3 and 4. In Chapter 3, we will consider privacy-preserving set interesection, a modification of a surveillance technique used by the FBI [1] and the NSA [49] to support preserving the privacy of all users except the actual subjects of an investigation. We will present a protocol to achieve privacy-preserving set intersection, and then discuss our experimental results with an implementation of this protocol.

Chapter 4 will discuss contact chaining, an investigative tool that examines connections in a social graph to attempt to discover other members of a known suspect's criminal or terrorist organization [15]. We will present a protocol for privacy-preserving chaining, which can be used in connection with our protocol of Chapter 4. We will also consider the running time of the privacy-preserving contact chaining protocol run on a sample social graph.

In Chapters 5 and 6, we turn our attention to anonymity protocols. We will discuss in Chapter 5 a vulnerability in the currently deployed implementation of Tor, which could allow a malicious user to game the system measuring relays in the Tor network and potentially associate users with their traffic. To resolve the vulnerability, we offer a new measurement system called Peerflow.

An alternative anonymous communication system to Tor is Verdict, originally proposed in [13]. In Chapter 6 we, for the first time, formally specify the Verdict protocol and rigorously prove how it uses a "dining cryptographers"-based protocol [10]

to guarantee anonymity for its users, integrity of messages sent, and accountability for anyone violating the protocol.

Finally, we conclude by offering some open questions and directions for future research in Chapter 7.

Chapter 2

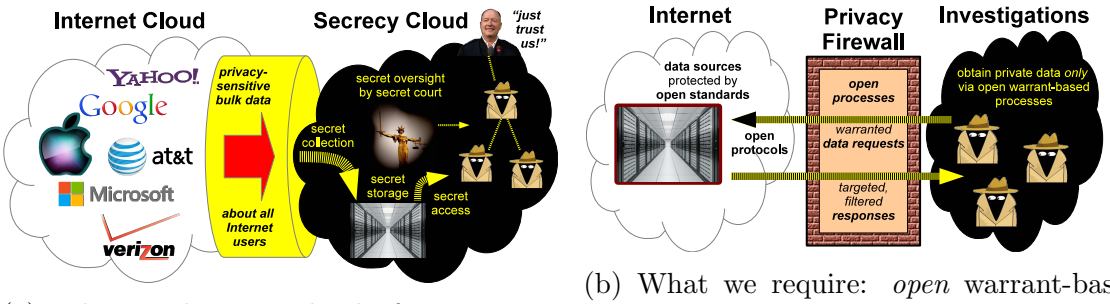
Privacy-Preserving Surveillance and Openness Principle

This chapter outlines several principles that we believe should govern electronic surveillance. We start with a basic principle stating that processes that use private data in bulk must be *open*, and we then outline several related properties that we expect such open processes to have. Finally, we summarize how these principles might be applied in the case of "set-intersection warrants."

This research was carried out in collaboration with Bryan Ford and Joan Feigenbaum, and the material in this chapter was presented in preliminary form in [45].

2.1 Open Processes for Law Enforcement

A basic tenet of democratic society is that law enforcement must follow *open processes*: procedures laid out in public law and subject to debate and revision through deliberation. Police need not disclose *whom* they may suspect of a particular crime or other details of an ongoing investigation, but their investigation must neverthe-



(a) What we have: a cloud of secret mass surveillance processes

(b) What we require: *open* warrant-based processes for lawful electronic surveillance, creating a "privacy firewall."

Figure 2.1: Secret versus open electronic surveillance processes

less follow rules and procedures established in *open* law books that everyone has a right to know and understand. And it is accepted that searching a person's home or personal records requires a narrowly targeted and properly authorized warrant based on probable cause.

We wish to formulate an openness principle for electronic surveillance that distinguishes between two classes of Internet users. A *targeted user* is one who is under suspicion and is the subject of a properly authorized warrant. All others are *untargeted users* – the vast majority of Internet users (and cell-phone users and users of any general-purpose, mass-communication system).

Just as search-warrant processes in free societies are grounded in open law, we believe that any "bulk" electronic-surveillance process that ingests, searches, or otherwise touches private[1] data of untargeted users must likewise be an open process. We refer to processes that are not open, public, and unclassified as *secret processes*, and we seek to limit their use (while recognizing that there are circumstances in which they may be needed). Once law enforcement has legitimately employed an open process to identify, target, and obtain information about an Internet user suspected of a crime, however, it may potentially subject that targeted user's data to

[1]Rigorous definitions of the term "private" are the subject of extensive study in computer security, law, philosophy, and many other fields. They are beyond the scope of this dissertation.

the full range of secret analysis tools and techniques in its arsenal.

One of the key reasons the NSA's mass-surveillance activities disclosed by Snowden are so troubling is that they tap into "bulk" data and metadata about untargeted users and ingest these private bulk data into secret processes that are codified only in secret FISA law and are subject only to secret oversight and accountability procedures (Figure 2.1a). In short, the public must simply "trust" the US government's evidence-free assertions that its mass ingestion and secret processing of privacy-sensitive data are (secretly) lawful and subject to adequate (secret) privacy protections and effective (secret) oversight. We cannot remotely envision the framers of the US Constitution being comfortable with such blind faith in secret mass-surveillance processes of this nature.

We therefore propose that a basic openness principle, comprising two main planks, should govern electronic-surveillance processes in a modern democracy:

I Any surveillance or law-enforcement process that obtains or uses private information about untargeted users shall be an open, public, unclassified process.

II Any secret surveillance or law-enforcement processes shall use only:

(a) public information, and

(b) private information about targeted users obtained under authorized warrants via open surveillance processes.

We view this openness principle as demanding that an open *privacy firewall* be placed in the path of private information flowing from the Internet to law enforcement (Figure 2.1b). Processes that search or ingest private data of untargeted users "through the firewall" must be open processes, but, once a user is targeted by a legitimate warrant and his data have been acquired via open processes, these data

collected about that targeted user may potentially be subject to secret investigative processes.

Openness conceived in this manner may sound incompatible with the requirement that government agencies be able to keep secret the targets and details of active investigations, but it is not. Using appropriate security technology, a data-collection or surveillance *process* used in an investigation may be made fully public without revealing the *content* of any particular investigation.

Our focus here is on general electronic surveillance principles for law enforcement purposes, independent of any particular government or agency. The hot-button case of the NSA is complicated by the fact that the NSA was founded as a foreign-intelligence agency but has acquired *de facto* characteristics of law-enforcement agencies by: (a) increasingly serving to support and feed surveillance data to law-enforcement agencies such as the FBI and the DEA; (b) collecting and storing both US and non-US surveillance data alike, even if internal "searches" are allowed only on "non-US persons"; (c) being increasingly employed not just against wartime adversaries but against citizens of peaceful, allied, democratic states, who common sense dictates should have protection against "unreasonable search and seizure" regardless of the letter of US law [29]. To whatever extent the NSA or any government agency behaves like a domestic or international law-enforcement agency, we believe the above openness principle should apply.

2.2 Mass Surveillance

How should this openness principle be applied to *mass-surveillance* processes, *i.e.*, processes such as the cell-phone records-collection program that have the potential to collect or use *all* data in a particular category about *all* users? (As currently

implemented, the cell-phone records-collection program realizes this potential [18,26], but we do not think it should.) We refer to data sets collected and used in mass surveillance as *bulk data sets*.

We identify four particular "sub-principles" that we believe should apply to mass-surveillance processes:

Division of trust: No single agency or branch of government should have either the authority *or the technical means* to compromise the privacy of bulk data about untargeted users. Mass-surveillance processes must require the sign-off, oversight, and active participation of multiple independent authorities representing each branch of government.

Enforced scope limiting: Surveillance processes must incorporate scope-limiting mechanisms ensuring that no particular warranted-surveillance activity captures data from an overly broad group of users. For example, each warrant might have a specified limit on the number of users whose data may be touched by the warrant-authorized process.

Sealing time and notification: Surveillance processes that capture privacy-sensitive user data must impose a limit on the length of time that the users in question may be kept ignorant of the fact that their data were captured. After this time has expired, the process must ensure that the users are notified of the data access and given means to investigate the justification and/or obtain recompense for any unjust effects of the investigation. Higher levels of authority should be required to authorize longer sealing times. No level of authority should permit indefinite sealing times (even indirectly, on an "installment" basis).

Accountability: Surveillance processes must incorporate accounting mechanisms that enable all three branches of government, as well as civilian participants, to maintain and safely disclose relevant statistics on how frequently and extensively

warranted-access mechanisms are used, *e.g.*, number of warrants per month of a given type, maximum number of individuals affected under any warrant, total number of individuals affected by all warrants in one month, or maximum secrecy period applied to any outstanding warrant in one month.

2.3 Case Study: Intersection Warrants Using Cell-Tower Dumps

Given a properly authorized warrant, we wish to enable law-enforcement agencies to target not just *known* users (those whose cell-phone numbers they already have and are covered by the warrant) but also *unknown* users (in our case, those whose cell-phone numbers they do not have but may be able to discover by *intersecting* several relevant cell-tower dumps). It may appear nonsensical to describe a user as both "unknown" and lawfully "targeted," but it is not. We may view such an *intersection warrant* as a type of "John Doe" warrant [5]: one in which the names or phone numbers of the person(s) of interest are unknown, but for which relevant times and locations are known, and for which there is sufficient evidence to convince a judge that there is probable cause to believe the given times and locations uniquely identify the unknown person(s) who committed a crime.

For example, the FBI caught the High Country Bandits [1] by intersecting three cell-tower dumps, representing the sets of cell-phone numbers that had been used near three different bank-robbery sites at the times of the robberies. In total, these dumps contained 150,000 cell-phone numbers, but their intersection contained only one: that of a High Country Bandit. Similarly, the NSA's CO-TRAVELER program [49] searches for unknown associates of known surveillance targets by first intersecting cell-tower dumps from times and locations at which a particular known

target appeared and then interpreting the intersection as the set of cell-phone numbers of people who may be "traveling with" the known target.

In Sections 3.1 and 3.2, we present and evaluate a protocol that computes the intersection of cell-tower dumps and obeys the principles articulated above. This is a natural test case for us for at least two reasons. First, intersections of cell-tower dumps have proven useful in catching criminals; this distinguishes them from many of the other surveillance activities featured in the Snowden revelations, the practical utility of which is at best unclear. Second, privacy-preserving set intersection is a well-studied, mature, and practical technology [24, 39, 55].

Note that our protocol is not specific to cell-tower dumps and could also be used to query other "time-and-place" metadata collections in an open, lawful manner.

Chapter 3

Privacy-Preserving Set Intersection

3.1 Lawful Intersection Attacks

This section first outlines the assumptions and principals involved in our lawful intersection-warrant protocol, then describes the operation of the protocol, and finally summarizes its key security properties.

This research was carried out in collaboration with Bryan Ford and Joan Feigenbaum, and the material in this chapter was presented in preliminary form in [45].

3.1.1 Principals

Our model for lawful intersection attack involves the following three types of principals. For simplicity, we assume here that these principals will participate in an intersection-attack mechanism in an honest-but-curious way. That is, they will not attempt to violate the rules of the mechanism, but they may use their own views of

all data they see to acquire additional information.[1]

Sources: entities that produce metadata records embodying information of the form, "user X was observed to be near location Y at time Z." The obvious examples are phone companies whose cell towers produce logs of the users who appeared in the vicinity of a given cell tower at a given time, but our model extends to other producers of metadata of this general form.

Repository: any entity tasked with storing metadata for surveillance or law-enforcement purposes. This may be the phone companies that produced the records (*i.e.*, the same as the metadata sources), a government agency, or some specialized independent agency. While "who stores the data" is an important question in general, it is orthogonal to our goals, and our model is agnostic with respect to its answer.

Agencies: a set of *multiple* independent but cooperating government agencies across whom our model divides surveillance authority. While our model is formally agnostic with respect to the number or specific natures of the authorities across whom trust is divided, we will use the US's 3-branch constitutional model as a concrete example, in which it might be appropriate to divide surveillance authority across three agencies:

- The **Executive Agency** represents the executive branch and is responsible for *requesting* surveillance warrants – *e.g.*, an agency like the NSA or FBI.

- The **Judicial Agency** represents the judicial branch and is responsible for *authorizing* requested warrants, after verifying independently that they are legally justified and suitably scoped.

- The **Legislative Agency** reports to the legislative branch and is responsi-

[1] This assumption could be relaxed significantly by requiring all principals to produce zero-knowledge correctness proofs of their intermediate results, using standard and well-known techniques. We leave these details to future work, however, and we would still need to assume the correctness of the original inputs – *e.g.*, logged phone numbers.

ble for ensuring that accurate and sufficiently detailed data are gathered and regularly reported to Congress on how and to what extent these surveillance capabilities are employed.

3.1.2 Lawful Set-Intersection Protocol

The lawful set intersection protocol we present is similar in structure to the protocol of Vaidya and Clifton [55].

Our protocol is built on two encryption schemes: ElGamal [21, 54], a randomized encryption scheme, and Pohlig-Hellman [44], a deterministic encryption scheme. Both of these cryptosystems are also *commutative*.

We use the randomized, public-key ElGamal encryption scheme for long-term protection of stored data. Each agency, or "participant," in the protocol needs an ElGamal key pair, the public key for which is known to the sources of private information. We use randomized encryption for data storage so that two ciphertexts representing the same piece of metadata will not necessarily be identical. This prevents the repositories or the participants from learning anything about the ciphertext sets - in effect, preventing them from performing the intersection attack without the cooperation of other participants.

In order to enable intersection at the appropriate point in the protocol, we use the deterministic, symmetric-key Pohlig-Hellman encryption scheme. The participants in the protocol use Pohlig-Hellman to blind the data prior to intersection. Short-term Pohlig-Hellman keys are generated by each participant during the protocol execution and discarded at the execution's end. Because Pohlig-Hellman is commutative and deterministic, there is a one-to-one correspondence between data items and their encryptions under any fixed set of Pohlig-Hellman keys, regardless of the order in which those keys are applied. We rely on this property to allow the intersection to

proceed.

The ElGamal and Pohlig-Hellman encryption schemes are both commutative. A commutative encryption scheme has the property that a message encrypted sequentially under multiple encryption keys can be decrypted by applying the corresponding decryption keys in any order. not only commutative but mutually commutative - that is, a message encrypted under a combination of encryption keys from the two cryptosystems can still be decrypted by the corresponding decryption keys, again regardless of order. For example, a ciphertext encrypted under a single ElGamal key could next be encrypted under a Pohlig-Hellman key to form a "mixed encryption" ciphertext. This ciphertext can then decrypted with the ElGamal decryption key to produce a valid Pohlig-Hellman ciphertext. As long as at least one of the keys used to encrypt a ciphertext is a key from the randomized ElGamal cryptosystem, the mixed encryption is randomized, and direct comparison between ciphertexts cannot be performed.

Each participant's input is its ElGamal private key and a set of data that has been encrypted under the ElGamal public keys of all agencies. The agencies do not generate these sets – rather, they are distributed to the agencies by the repositories. If there are n agencies, they are given numbers 1 through n so that, when the m^{th} agency is done acting on a set of data, it can pass the set on to the $(m+1)^{th}$, and it can receive new sets from the $(m-1)^{th}$.

Assuming all agencies execute the protocol with honest-but-curious behavior, the protocol's output for each participant will be the intersection of all sets. Optionally, each agency may also supply a threshold limiting the size of the intersection it is willing to reveal. If the size of the intersection of all sets would be above any agency's threshold, no agency will learn anything except the cardinality of the intersection, which agency's threshold was violated, and some intermediate values discussed in

Section 3.1.3. The protocol runs as follows:

Initialize. Each agency first generates a temporary Pohlig-Hellman key to be used only during this execution of the protocol and then discarded. The first agency then obtains the ElGamal-encrypted sets to be intersected from the Repository.

Phase 1. Each agency in turn uses its ElGamal private key to remove a layer of ElGamal encryption from each item in each set to be intersected; it then adds a layer of Pohlig-Hellman encryption to each item, using the temporary key it generated in the Initialize step. The agency then randomly shuffles each encrypted set independently, while keeping the sets separate, and forwards the sets to the next agency. The phase is complete when agency n decrypts the final layer of ElGamal encryption from all items in all sets, leaving all sets encrypted under every agency's Pohlig-Hellman keys only.

Phase 2. Agency n broadcasts the resulting Pohlig-Hellman-encrypted data sets to all other participants. Each participant then computes the desired intersection: *i.e.*, the encrypted elements that appear in all sets – or, more generally, the elements that appear in some threshold number of the sets as defined by the intersection warrant. Because Pohlig-Hellman is a commutative and deterministic encryption scheme, two identical data items will have identical encryptions at this stage, making computation of the intersection trivial despite the encryption.

Phase 3. If any agency sees that the number of distinct items (*e.g.*, phone numbers) appearing in the resulting set intersection is above a warrant-specified limit on the number of individuals the warrant is permitted to target, the agency deletes its Pohlig-Hellman key, sends a message to all other agencies, and refuses to continue with the protocol. (The agency requesting the warrant might then be required to produce a new, more narrowly targeted warrant and try again.)

Phase 4. If the intersection's cardinality meets the requirements of the warrant,

then the agencies collectively decrypt the items in the intersection. As in Phase 1, each agency in turn uses its Pohlig-Hellman key to decrypt each element of the intersection set, shuffles the intersection set, then forwards it to the next agency. The phase completes when when agency n decrypts the last layer of Pohlig-Hellman encryption and forwards the plaintext result to the other agencies.

For simplicity, we describe Phases 1 and 4 above as strict "cascades," each agency processing the full data set before passing it to the next. A simple performance optimization our prototype implements is for different input sets to start at different agencies – i.e., to start and end at different "points" around a circle – thus spreading computational load and increasing parallelism. This is only one of many potential optimizations, however.

3.1.3 Protocol Properties

We now analyze our cell-tower-dump intersection mechanism with respect to our openness principle for mass-surveillance processes. We accomplish division of trust by having all data be encrypted in advance with the public keys of the agencies that request, authorize, or oversee the surveillance. Without participation of all of these agencies, the data cannot be decrypted – even if the Repository is compromised, for example – and no unauthorized surveillance can be performed unilaterally.

The protocol also provides the means to enforce a limited scope of investigation. The sizes of all sets and intersections are visible to all participants in Phase 3; so the warrant can specify a limit on the number of users whose data may be revealed. If it does, and the size of the intersection is above this limit, the protocol participants should stop *before* any metadata records are revealed in cleartext. Any or all of the agencies can take responsibility for this, because even one agency halting and deleting its Pohlig-Hellman key prevents decryption of the metadata.

If the protocol completes, it gives the same output to each participant. This makes it easy to notify users whose data were viewed after some sealing time and to maintain statistics for the purpose of accountability. These processes are beyond the scope of the intersection protocol, but one of the participants in the protocol can be responsible for maintaining them. The other participants can audit those processes, since they also know which uses should be notified.

Finally, this process protects the privacy of untargeted users. The only information leaked apart from the output are the *sizes* of intersections of any two or more sets involved in the protocol. (This property is proven in [55] for a protocol using the same structure as ours; the proof generalizes straightforwardly to our case.)

This small information leakage reveals how many users appear in multiple sets but does not reveal any specific user identities or metadata other than those in the requested intersection. Because only aggregate properties of the sets are leaked, we feel this leak should not represent a major privacy issue – and when a query fails because of an empty or too-large intersection, the leaked statistics may help the agency that requested the warrant formulate a revised request for a better scoped warrant.

3.2 Implementation and Evaluation

This section presents the results of our preliminary experiments with the lawful set-intersection protocol of Section 3.1. Recall that each network node that participates in this protocol acts on behalf of an "agency," in the terminology of Section 3.1.1. It is this set of agencies across whom trust, *e.g.*, the power to authorize an intersection attack, is divided in our model. Because the public keys of *all* agencies are used to encrypt the data stored in the Repository, each agency effectively uses its private

key to "authorize" the selection and decryption of results responsive to a particular intersection warrant.

3.2.1 Prototype Implementation

Our implementation of the lawful set-intersection protocol is written in Java and available on GitHub.[2] The prototype does not use any external libraries for cryptography beyond Java's standard BigInteger class.

The program is run on multiple servers. The servers connect to each other over TCP sockets, and, for simplicity, we use a directed cycle as the connection graph. Each server sends data only to the next server in the cycle and receives data only from the previous server in the cycle. Each participant takes one set of encrypted data and a 1024-bit ElGamal private key as input. The data must have been sequentially encrypted under all participants' public keys before it can be used in the protocol; because this encryption is done offline and in advance, the protocol itself does not require access to public keys.

To test the protocol, we ran it many times on data sets of various sizes. We used three nodes on the Yale CS Cloud, each running 8 compute threads to process the encryption and decryption of data.

The tests were all run using only these three nodes, each node with one data set. We expect the running time would increase considerably as a function of the number of participants, but three is a natural number of nodes in this context, representing a distribution of authority across three branches of government (Section 3.1.1).

[2] `https://github.com/DeDiS/Surveillance`

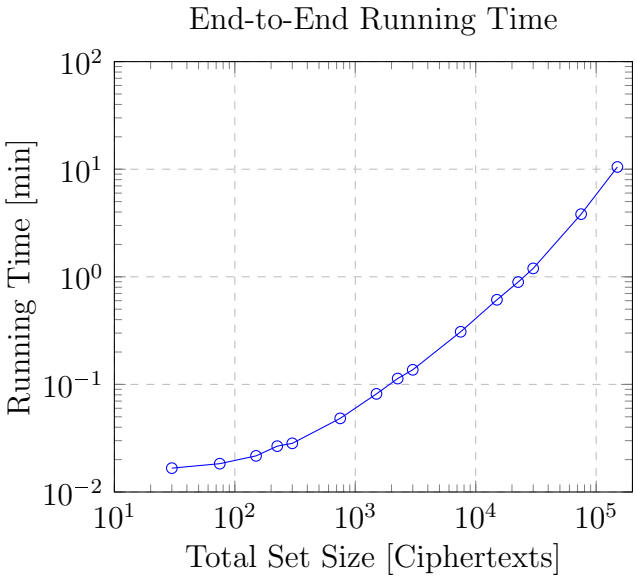


Figure 3.1: Lawful Intersection Performance

3.2.2 Query Efficiency

Prior to execution, we randomly generated data sets for each trial for each node. We ran the protocol 10 times each with different-sized data sets, ranging from 10 items per set to 50,000 items per set. In each trial, we measured the amount of data each node transmitted upstream during the protocol and the total end-to-end time of the protocol, measured from the start of the protocol's execution to the production of output. After running each test 10 times, we averaged the results; these averages are presented in Figure 3.1.

In the High Country Bandits case, the FBI processed information from about 150,000 users total [1]. Our largest test, with 50,000 data items per set, tested our protocol's efficiency with an equally large amount of data. The average amount of time needed to run the protocol in this experiment was 629.4 seconds, just over 10 minutes.

These tests were run with an intersection size of three. We also tested these benchmarks with an intersection size of 10 and found that the average times did not

change by more than one second in any case and that the data sent per node always increased by 3 KB.

Our results indicate that the amount of data sent over the network and end-to-end time both increase linearly with the size of the data sets, which is what we would expect from this protocol.

Further tests showed that total data sent and total CPU usage across participants were not affected if the data were concentrated in one or two sets, as opposed to being spread equally over all three sets. However, we found that the end-to-end delay can increase by up to a factor of two if the data are spread out. This result is unsurprising, because unbalanced sets render less effective the optimization mentioned at the end of Section 3.1.2, wasting time while the small-set-input participants idle, waiting for data to be sent by large-set-input participants.

Chapter 4

Privacy-Preserving Contact Chaining

Contact chaining [15] is a form of government surveillance with which it is deceptively easy to expose many innocent, untargeted users to government scrutiny. In this chapter, we propose a privacy-preserving surveillance protocol to obtain relevant information that would be discovered using a contact chaining search, without violating the privacy of untargeted users.

The work reported in this chapter was done in collaboration with Bryan Ford and Joan Feigenbaum.

4.1 Lawful Contact Chaining

The goal of contact chaining is to use information about social connections between identities, such as records of phone calls between one number and another, to identify members of a criminal organization or terrorist group. Starting with one or more suspects whose identities are known, the government aims to consider contacts of

those suspects. These can be *direct contacts*, such as two people who spoke on the phone, or *extended contacts*, such as two people connected by a chain of two or more phone calls. For example, suppose Alice calls Bob, Bob calls Charlie, but Charlie and Alice have not called each other. Then Alice and Bob are direct contacts (as are Bob and Charlie), but Alice and Charlie are extended contacts. We may also say that Alice and Charlie are at distance 2 in the communication graph (because the smallest number of phone calls that connect Alice to Charlie is 2).

Without mechanisms to preserve privacy, a contact chaining search can collect a surprisingly large group of users' information. For example, if the average cell phone user contacts 30 individuals within the period of the investigation, a contact chaining search out to distance 3 would capture 27,000 users on average - or many more if a heavy phone user is swept up by the search. With such a large group, it is assured that the vast majority of contacts will not be the targeted collaborators of the primary suspect in the investigation. This is a large and unnecessary intrusion of privacy. These untargeted users may nevertheless face unwarranted government scrutiny, intrusive investigation, or a risk that their sensitive communications histories may be leaked accidentally.

Despite this risk, we recognize the potential law-enforcement value of information about social connections between targeted invidivuals. Therefore, we propose a *lawful contact chaining protocol*. This protocol permits multiple government agencies working together to provide oversight and accountability, satisfying the principles laid out in Chapter 2. Our protocol focuses on the case in which the government seeks information from multiple telecommunications providers about the communication graph formed by phone calls and text messages. Using this protocol, the agencies can retrieve an encrypted set of user data from multiple telecoms, each of which holds only part of a larger communication graph. This encrypted set contains the

identities of users within a certain distance of a target, but the identities cannot be decrypted unless the agencies cooperate. Under the lawful processes we propose, this cooperation would take the form of an intersecton with other encrypted sets of data, using the protocol from Section 3.1.2. These sets can come from privacy-preserving contact chaining, from cell tower dumps, or from other sources of information about suspects. Importantly, while any set may contain encrypted data about many untargeted users, very few users will appear in *all* the sets, and those few users will be suitable targets for further lawful investigation.

The same principles of oversight and accountability provided by multiple government agencies can apply to contact chaining searches in other types of communication graphs, such as the social network graph of Twitter or Facebook. These cases do not require our protocol, however, since if one provider knows the entire communication graph, it can compute the output of the protocol without any interactivity needed.

4.2 Protocols For Privacy-Preserving Contact Chaining

4.2.1 Inputs and Parties to the Protocol

There are two types of parties in this protocol: Telecommunications companies (telecoms) and government agencies interested in performing lawful contact-chaining (agencies). The protocol computes a function of all parties' data.

The telecoms jointly hold an undirected communication graph $G = (V, E)$. Each telecom knows only a subset of the edges E. V contains vertices labeled with the phone numbers they represent, and E contains an edge between a and b if and only if phone number a has contacted phone number b or vice versa within some window of

time. Each phone number v is served by exactly one telecom. We assume telecoms know which telecom serves which phone number. Each telecom keeps records of all phone calls made by phones they serve, including calls made to phone numbers served by other telecoms. The subgraph known by telecom T is $G_T = (V, E_T)$ where E_T is the set of edges (a, b) such that a or b is a phone number served by T. Henceforth, for any phone number a, let $T(a)$ be the telecom that serves a.

The agencies must each hold a copy of a *warrant* in order to perform this protocol. A warrant is a triplet (x, k, d). The variable x is a target phone number. We assume, since x belongs to a user targeted by the agencies, that they also know which telecom serves x. The variable k is a (small) distance from x, the distance at which the agencies wish to consider chained contacts. For example, if $k = 2$, then the agencies only wish to consider users at most 2 phone calls away from their person (or phone number) of interest x. Choosing a small limit is important to limiting the scope of the investigation. However, many users' information might still be captured if some phone numbers have very many contacts. Suppose the target x calls the most popular pizza place in town. Now everyone else who has recently called that pizza place is at a distance 2 to x.

We can assume that business phone numbers have many more contacts than personal phone numbers do. In most cases, knowing that two individuals have contacted the same business phone number does not indicate that those individuals have a personal relationship. Therefore, the warrant also includes d, an upper bound on the degree of users the agencies are willing to "chain" through. If a phone number has more than d contacts, then the agencies do not consider paths to other users through that phone number in their search. The agencies disregard d for the initial target x, however. The high-degree users themselves will also be present in the agencies' outputs.

This provides a reasonable limit to the scope of the investigation and hides what are very likely to be untargeted users from the government. In the uncommon scenario where a business number with many contacts also functions as a front or hub for a criminal organization to be revealed, the government is still able to conduct further investigation on it, perhaps even beginning a new contact-chaining search with that number as the initial target.

We specify the protocol in full in Sections 4.2.4 and 4.2.5.

4.2.2 Security Assumptions

We make a few assumptions about existing cryptographic infrastructure. All telecoms and agencies must have a public encryption key known to all other parties to the protocol and a private decryption key. For the purpose of interoperability with lawful intersection, agencies' keys must be for a commutative cryptosystem (i.e. ElGamal). The parties must also each have private signing keys and public verification keys.

In the protocol below, we refer to "the agencies" sending messages to one or more telecoms. Exactly which agency transmits messages to the telecoms is not important to our protocol, but a telecom will disregard any message not accompanied by signatures from every agency. One simple topology is for a single agency to handle all direct communication with telecoms and with other agencies, forwarding reponses from the telecoms on to the other agencies and signatures on agency messages to the telecoms.

Our protocol preserves the privacy of untargeted users as long as all parties execute the protocol in an honest-but-curious manner, all of the government agencies do not collude together, and no telecoms collude with government agencies. A colluding group containing all agencies would be equivalent to the current situation n which

the government does not provide meaningful accountability of its own surveillance activities; what we propose is a replacement for this situation, but it does require the government to follow its own laws, once set. A telecom colluding with a government agency would amount to sending that agency free information about its users, or submitting incorrect information to the protocol. But telecoms have no business purpose to deviate from the protocol and risk legal action. In practice, existing legal tools allow law enforcement agencies to gather information about the phone history of a suspect with a valid warrant, but such information cannot generally be used for further contact chaining.

In Chapter 7, we discuss potental ways in which our honest-but-curious assumption might be relaxed.

4.2.3 Desired Outputs and Privacy Properties

The goal of the protocol is for the agencies to obtain a set of ciphertexts, each of which is the encryption of a phone number v such that the distance in the communication graph from v to the targeted phone number x is at most k. The set should not contain encryptions of numbers v such that each path from x to v of length at most k contains an intermediate vertex of degree greater than d. Here the "intermediate" vertices in a path are all vertices except the endpoints x and v.

Every phone number in this set must be encrypted with each of the agencies' public ElGamal keys. The agencies should all have the same output.

The telecoms should not learn the agency's output. Instead, each telecom's output should contain only a list of which of the phone numbers it serves were sent to the government agencies. This allows the telecoms to play an additional accountability role in this protocol. The government may have an interest in keeping the telecoms from knowing which of their clients were surveilled; we discuss this in section 4.3.3.

With the encryptions of these phone numbers, the agencies can then act as appropriate to further investigate them. In particular, the encrypted set of phone numbers can be used as an input into a lawful set intersection protocol.

Below, we present two versions of our protocol. In the first version, the agencies and telecoms learn some additional information. Specifically, the agencies learn the provider of each phone number in the encrypted set, and the distance from x of each encrypted phone number. Each telecom learns which of the phone numbers it serves appear in the agencies' output, as well as the distance of each of those phone numbers from the target phone number x.

In section 4.2.5, we will present a second version of the protocol in which the agency *does not* learn which telecom serves which encrypted phone number.

As long as our security assumptions for this protocol hold, the agencies collectively learn *no* information about the edge set E except what is implied by the output. Furthermore, the agencies cannot learn any of the phone numbers that appear in encrypted form in the output (unless implied by the size of the encrypted output and the leaked service information), nor can agencies cause a phone number not within distance k of target x to appear in the output, even in encrypted form.

4.2.4 Ownership-Revealing Lawful Contact-Chaining Protocol

The protocol below amounts to a distributed breadth-first search of the communication graph run by the agencies making queries of the telecoms. However, all messages the agencies receive from the telecoms will be encrypted. They will know which message come from which telecoms, and will therefore know which telecoms serve which ciphertexts.

Let $\text{Enc}_T(m)$ be the encryption of message m under telecom T's public key. Call such an encryption a *telecom ciphertext*. Let $\text{Enc}_{\mathcal{A}}(m)$ be the encryption of m under the public keys of all agencies, and call such an encryption an *agency ciphertext*.

To manage the breadth-first search, the agencies (or at least the investigating agency) will maintain a queue $\mathbf{Q}$, containing vertices yet to explore. $\mathbf{Q}$ contains tuples for unexplored vertices a of the form $(\text{Enc}_{T(a)}(a), T(a), j)$. These tuples contain the telecom ciphertext for a, a record of which telecom owns a, and an integer j indicating the remaining distance out to which neighbors can be chained from a.

The agencies will represent their output in the form of a list $\mathbf{C}$, containing agency ciphertexts. Each telecom T will represent its output in the form of a list $\mathbf{L}_T$, listing plaintext users served by that telecom whose information the agencies requested.

The protocol is as follows:

1. The agencies start by agreeing upon a warrant (x, k, d), where x is the target phone number, k is a maximum distance, and d is an upper limit on the degree of vertices to chain through. They encrypt x under the public key of $T(x)$.

2. The agencies initialize a queue $\mathbf{Q}$. Initially, $\mathbf{Q}$ contains only the triple $(\text{Enc}_{T(x)}(x), T(x), k)$.

3. The agencies initialize the output list $\mathbf{C}$ to be empty.

4. Each telecom T initializes its output list $\mathbf{L}_T$ to be empty.

5. While $\mathbf{Q}$ is not empty, do the following:

   (a) The agencies dequeue $(\text{Enc}_{T(a)}(a), T(a), j)$ from $\mathbf{Q}$. They send the pair $(\text{Enc}_{T(a)}(a), j)$ to $T(a)$.

   (b) a's provider, $T(a)$, decrypts a from its telecom ciphertext. It adds a to $\mathbf{L}_T$.

(c) $T(a)$ encrypts a under the agencies' public keys, and sends $\text{Enc}_{\mathcal{A}}(a)$ to the agencies.

(d) If $j = 0$, $T(a)$ is done. Go to step 5g.

(e) Otherwise, $T(a)$ encrypts each neighbor b of a under the public key of $T(b)$, creating a telecom ciphertext for b.

(f) $T(a)$ sends the number of ciphertexts generated this way, $\deg(a)$, as well as all telecom ciphertexts generated in the previous step, to the agencies. $T(a)$ sends the ciphertexts in the form of pairs $(\text{Enc}_{T(b)}(b), T(b))$.

(g) The agencies add $\text{Enc}_{\mathcal{A}}(a)$ to $\mathbf{C}$.

(h) If $\deg(a) > d$ and $j \neq k$ (i.e. $a \neq x$), the agencies discard all telecom ciphertexts received for a's neighbors (i.e., agencies refuse to sign these ciphertexts in future steps of the protocol, and do not send them on to the telecoms).

(i) Otherwise, for each telecom ciphertext received, the agencies add $(\text{Enc}_{T(b)}(b), T(b), j-1)$ to $\mathbf{Q}$.

6. The agencies' final output is the list $\mathbf{C}$. Each telecom T's final output is $\mathbf{L}_T$.

For the sake of efficiency, it is worth noting that the inner loop can be executed many times in parallel, up to the point of completely emptying $\mathbf{Q}$ at the beginning of the loop. Many messages to the same telecom can also be batched and sent together, thereby reducing the number of signing and verifying operations so that they depend only on k and not on the size of the input or output.

4.2.5 Ownership-Hiding Lawful Contact-Chaining Protocol

The previous version of the protocol allows agencies to learn which telecoms own the phone numbers in its encrypted output. This subsection presents a modification of the previous version of the protocol, which uses a DC-nets-based *anonymity protocol* to hide this information from the agencies (except with respect to the initial target x).

An anonymity protocol is run by a number of parties, some of which have messages to send. At the end of the protocol, all parties must learn all messages sent, but no party other than the sender of any given message can learn which party sent that message. Dissent [12] and Verdict [13]) both satisfy our requirements; they are more powerful than we need, however, because we assume all telecoms are honest-but-curious. Given an anonymity protocol, we use it to to allow the correct telecom to respond anonymously in steps 5c and 5f in the protocol above. This removes the need for the agencies to know which telecom owns which ciphertext.

Now we can present the following modified protocol. This protocol uses the same data structures as in section 4.2.4, except that $\mathbf{Q}$ now contains pairs $(\mathrm{Enc}_{T(a)}(a), j)$ for unexplored vertices a (omitting the identity of $T(a)$.

1. The agencies start by agreeing upon a warrant (x, k, d), where x is the target phone number, k is a maximum distance, and d is an upper limit on the degree of vertices to chain through. They encrypt x under the public key of $T(x)$.

2. The agencies initialize a queue $\mathbf{Q}$. Initially, $\mathbf{Q}$ contains only the pair $(\mathrm{Enc}_{T(x)}(x), k)$.

3. The agencies initialize the output list $\mathbf{C}$ to be empty.

4. Each telecom T initializes its output lists $\mathbf{L}_T$ to be empty.

5. While $\mathbf{Q}$ is not empty, do the following:

   (a) The agencies dequeue a pair $(\mathrm{Enc}_{T(a)}(a), j)$ from $\mathbf{Q}$. They send the pair $(\mathrm{Enc}_{T(a)}(a), j)$ to all telecoms.

   (b) All telecoms attempt to decrypt $\mathrm{Enc}_T(a)(a)$ with their decryption keys. Only $T(a)$ will be able to do so. Other telecoms skip to step 5f.

   (c) $T(a)$ adds a to $\mathbf{L}_T$.

   (d) $T(a)$ encrypts a under the agencies' public keys, producing the agency ciphertext $\mathrm{Enc}_{\mathcal{A}}(a)$.

   (e) If $j > 0$, $T(a)$ encrypts each neighbor b of a under the public key of $T(b)$, creating a telecom ciphertext for b.

   (f) All parties to this protocol engage in the anonymity protocol. $T(a)$ sends an anonymous message consisting of the agency ciphertext it generated in step 5d; the set of telecom ciphertexts generated in step 5e, and $\deg(a)$, the number of telecom ciphertexts being sent. The agencies and all telecoms that could not decrypt $\mathrm{Enc}_{T(a)}(A)$ participate but send no anonymous message.

   (g) When the anonymity protocol is complete, the agencies receive all the ciphertexts. They add $\mathrm{Enc}_{\mathcal{A}}(a)$ to $\mathbf{C}$.

   (h) If $\deg(a) > d$ and $j \neq k$ (i.e. $a \neq x$), the agencies discard all telecom ciphertexts received for a's neighbors (i.e., agencies refuse to sign these ciphertexts in future steps of the protocol, and do not send them on to the telecoms).

   (i) Otherwise, for each telecom ciphertext received, the agencies add $(\mathrm{Enc}_{T(b)}(b), j-1)$ to $\mathbf{Q}$.

6. The agencies' final output is the list $\mathbf{C}$. Each telecom T's final output is $\mathbf{L}_T$.

The protocol replaces each query in the protocol of section 4.2.4 with broadcast of the telecom ciphertext to all telecoms, and replaces each response with a round of the anonymity protocol. This allows the telecom that owns each phone number to respond with appropriate information about the phone number, but shields the telecom's identity from the agencies (and incidentally from other telecoms).

As in the previous section, It should be noted that the agencies and telecoms need not handle one ciphertext at a time. The agencies can in principle dequeue all of $\mathbf{Q}$ in step 5a and broadcast all pending vertices to the telecoms. In step 5f, multiple telecoms can submit multiple messages to a single run of the anonymity protocol, with only those telecoms which were unable to decrypt any vertices submitting no message. The exact number of messages per instance of the anonymity protocol can be tuned for best efficiency.

4.3 Discussion of Lawful Contact-Chaining

We now discuss the correctness and privacy properties of both variants of our lawful contact-chaining protocol.

4.3.1 Correctness of Output

The agencies' outputs from the protocols in Sections 4.2.4 and 4.2.5 will be $\mathbf{C}$. $\mathbf{C}$ will contain agency ciphertexts of all phone numbers at most k phone calls away from x, considering only vertices of degree at most d. This is the desired output. C reveals nothing to any agencies unless they all provide their decryption keys. To continue the process of lawful investigation, the agencies should combine the output $\mathbf{C}$ with

other sets of potential suspects (such as from further runs of this protocol, or from cell tower dumps) in a lawful intersection protocol.

4.3.2 Privacy

Both versions of the protocol hide the identities of the chained contacts of x. They do allow the agencies to learn the distance from x of each ciphertext in their output, but these ciphertexts cannot be resolved to phone numbers without the cooperation of all agencies.

The protocol of section 4.2.4 allows the agencies to learn which telecoms owns which ciphertexts in $\mathbf{C}$. This may be a security concern, since some telecoms are relatively small, specialized, or localized to a particular country or region. If the agencies know that such a such a telecom owns an encrypted phone number, this will not allow them to identify the phone number itself, but might convince the agencies to subject that ciphertext to additional scrutiny, up to the point of decrypting it outside the context of lawful surveillance. This would still require the collusion of all agencies, however. Our revised protocol mitigates this concern. Assuming that the anonymity protocol used in section 4.2.5 does not allow its participants to learn who sends each message, then the revised protocol does not leak ciphertext ownership information.

The telecoms learn two types of information as part of the lawful contact chaining protocol. First, they learn the warrant. Second, they learn which of the phone numbers they serve have been captured (in encrypted form) by the protocol, and when they were captured. The telecoms might possibly be able to infer some extra information about G from observing when vertices they own are queried by the agencies, but only of a very limited form. For instance, an agency may serve two phone numbers, a and b, which the agencies query at distance 1 and 4 from x,

respectively . In that case, the telecom can infer that there exists a path in G of length 3 between a and b. The telecom does not learn which other phones are involved in that path, and is already aware of all paths of length 2 or less between phone numbers it serves. Therefore, this potential information leak is of little concern.

4.3.3 Hiding Information From Telecoms

In both versions of our protocol, the telecoms learn which of their numbers have been submitted to the agencies. They do not know which phone numbers the agencies will actually investigate after running the privacy-preserving set intersection protocol, but they do know which ones *could* be under investigation. Since many numbers *could* be investigated, this does not compromise the agencies' investigative power.

We may point out nevertheless that a modification of our protocol from 4.2.4 could allow the agencies to hide from the telecoms which of their clients is being surveilled. The telecoms would need to precompute agency ciphertexts for all of their client numbers, and telecom ciphertexts for all of their clients' contacts. With these precomputed databases, the telecoms could then use oblivious transfer to blindly serve the agencies' requests for information about their clients.

4.4 Performance of Privacy-Preserving Contact Chaining Protocol

We implemented the privacy-preserving contact chaining search protocol of 4.2.4 in Java and tested the implementation's running time, CPU time used, and data sent over the network. Below, we describe our implementation and its experimental setup, and then summarize our results.

4.4.1 Java Implementation

Our Java implementation uses the variant of our protocol in which the agencies completely exhaust the search queue $\mathbf{Q}$ each round, sending all queries at any given distance from x to the telecoms at once in batches. Ths variant allows for greater parallelism. All of the telecoms receive their batch of queries at the same time, and operate on those queries using eight parallel threads of computation.

We use 2048-bit DSA signatures, 2048-bit RSA encryption for the telecoms, and ElGamal encryption for the agencies' output to provide compatability with the lawful intersection protocol of Chapter 3.

Our Java program supports any number of agencies and telecoms, but we chose to run tests with three government agencies and four telecoms. Each agency and telecom has a dedicated server in our cloud testbed. As mentioned in Section 3.1.1, three is a reasonable choice for the number of agencies, corresponding to three branches of government. Four telecoms should cover most users in any given mobile phone market, and increasing the number of telecoms in our experiments only serves to decrease the protocol's total running time by splitting the same users over more servers.

4.4.2 Experimental Setup

For our underlying contact graph, we used an anonymized data set provided by [40] containing 1.6 million users from Pokec, a Slovakian social network. To replicate the multi-provider environment of the real telephone network, we assigned each user to one of four telecom servers. The telecoms were each given a different number of the users, in proportion to the subscriber base of the largest four telecoms in the world [42].

To experiment with differently sized output sets, we ran our protocol many times, varying x, k, and d. We chose a variety of different-degree starting targets x, varied the maximum path length k between 2 and 3, and varied d from 25 to 500. For each run, we measured the total running time of the protocol, the CPU time spent by the agencies and telecoms, and the amount of data sent over the network in total.

These results are important in evaluating how practical our lawful contact-chaining protocol would be it were put into practice by government agencies and telecoms. However, our data set is relatively small compared to the databases held by real telecommunications companies, and each company handles that data using different technologies. The absolute running time and CPU usage of executing this protocol could vary from telecom to telecom. Therefore, we also produced a implementation of the contact-chaining protocol which omits all cryptographic operations. This version of the protocol does not preserve the privacy of users. By comparing the performance of our lawful contact-chaining protocol with the zero-cryptography contact-chaining protocol, however, we can get a sense of the "cost" of privacy and accountability as compared to the practice of releasing plaintext data to government surveillance.

4.4.3 Results

Our implementation of lawful contact-chaining performed well. Our experiments showed a linear relationship between the number of ciphertexts in the output and the running time, CPU time, and data usage of the protocol. We display graphs of our recorded data in Figure 4.1. Taking the average of all cases with $d > 25$, the telecoms used 58.2 ms of CPU time per ciphertext. The agencies used, again in the average case, 2.0 ms of CPU time per ciphertext. Note that these times are the sums taken over all telecoms and all agencies respectively. Because the agencies have do very little cryptography in this protocol, we focus on the telecoms' CPU time in our

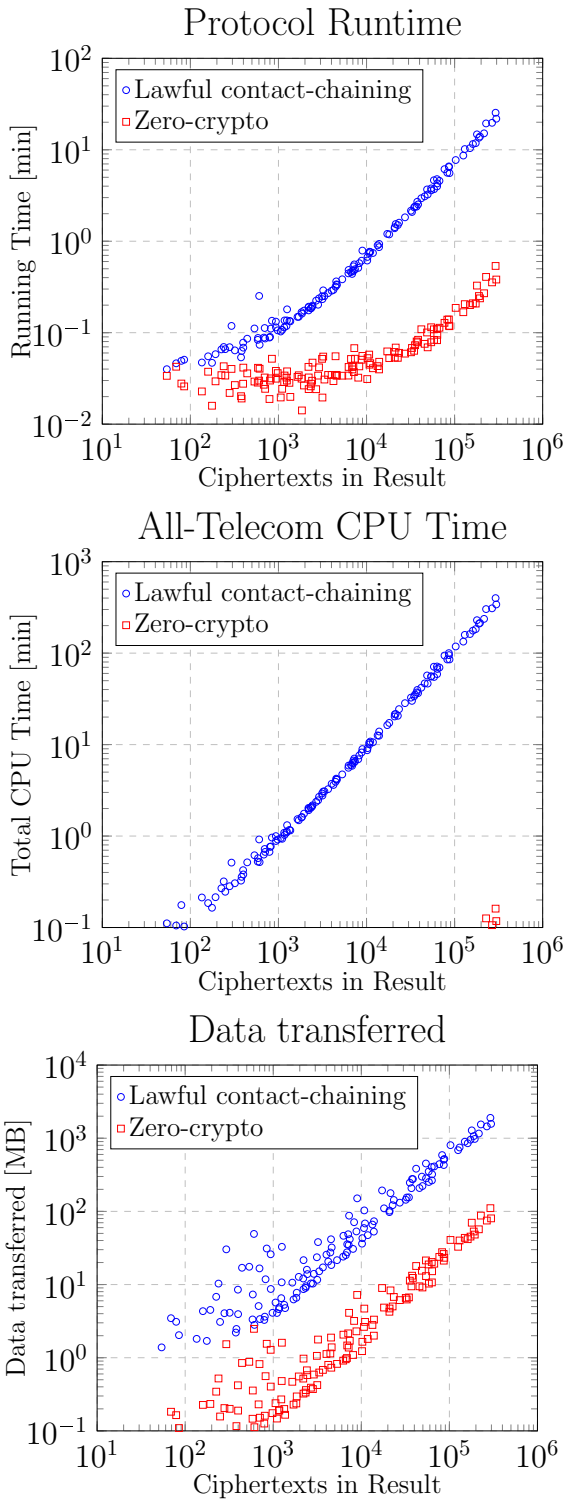


Figure 4.1: Performance of Lawful Contact-Chaining

evaluation.

We found that our protocol was able to process, in the average case, 197.4 ciphertexts per second. To return to our example from earlier of a network with an average of 30 contacts per user, a lawful contact-chaining search with $k = 2$ would have 900 users in the output, and a search with $k = 3$ would have 27,000 users in the output. To compare these times to some of our acutal experiments, we found that a search that returned 937 ciphertexts took 6.86 seconds to run, and a search that returned 27,338 ciphertexts took 109.55 seconds to run. To provide another point of comparison, consider the "High Country Bandits" case mentioned in Section 2.3, in which the FBI performed an intersection of 150,000 phone number to help solve a series of bank robberies. In one of our experiments with lawful contact chaining, we find that a similarly sized data set of 149,535 ciphertexts took 625.08 seconds - 10.4 minutes - to compile with our protocol. Given the context of a criminal investigation, we feel these running times are quite reasonable.

The zero-cryptography version of our program ran, predictably, more quickly than the lawful privacy-preserving version. The total CPU time across all telecoms needed for our zero-crypto implementation never rose above ten seconds, even in the largest cases. This result allows us to disambiguate the cost of *information retrieval* from *privacy protection*. The linear relationship between the size of the encrypted user data set and the performance in terms of running time, CPU time, and network data usage of the protocol all remain even when we subrtract out the time to run all non-cryptgraphic parts of the protocol. We therefore conclude that, even given the potental database operations real telecoms would have to contend with, the cost of adding privacy-preservation to the contact-chaining protocol will remain reasonable.

Chapter 5

Peerflow: Secure Load Balancing in Tor

We now turn our attention to anonymous communication. Anonymous communication protocols allow users to hide their identity, not only from the other parties they communicate with, but ideally also from third parties who might attempt to associate messages on the network with their senders. This allows users to avoid having their private information linked back to them and used in malicious or insecure ways.

Tor [19] is a popular anonymous-communication network. It consists of over 7000 volunteer relays carrying the traffic of over 2 million users daily at over 60Gib/s on average [53]. To balance this large traffic load over a diverse relay population, Tor estimates relay bandwidth using both self measurements and external measurements and then directs clients to use relays in proportion to the relays' bandwidth estimates. This load-balancing system is an attractive target for attack because the relays that carry a client's connection are able to learn sensitive properties about that connection, such as the client's identity. If an adversary controls enough of those relays, he can *deanonymize* the connection [36]. In this chapter, we discuss

how one of the systems used to manage these relays, TorFlow, could be exploited by an adversary, and we present a more secure, alternative system called PeerFlow.

The work reported in this chapter was done in collaboration with Aaron Johnson, Rob Jansen, Nicholas Hopper, and Paul Syverson.

5.1 Relay Measurement with TorFlow

Tor designed its current relay-measurement system, TorFlow [2, 43], to avoid relying entirely on self-reported measurements [4] and thereby improve performance and security. TorFlow implements "bandwidth scanning", in which measurement authorities create connections through relays to measure their bandwidth. Researchers have observed [6, 51] that in this system a malicious relay can increase its apparent bandwidth. We confirm this observation by implementing the attacks and experimentally testing them. Our results show that an adversary can obtain 489 times more client traffic than he should. An adversary with 1% of the network bandwidth could have 89% of client traffic directed to him, which he can then attack using deanonymization techniques that don't require that actually send or receive all of it [4].

The main alternative to TorFlow is the EigenSpeed system of Snader and Borisov [46–48]. In EigenSpeed, each Tor relay measures the speeds of its connections to other Tor relays and reports them to an authority, who applies Principal Component Analysis (PCA) to produce bandwidth estimates. We show that EigenSpeed too is highly vulnerable to attack. We identify basic flaws in its measurement method and initialization process, and we describe and experimentally demonstrate fundamental flaws in using PCA to aggregate measurements. These flaws allow an adversary to either get most honest relays kicked out of the network or receive up to 420 times more client traffic.

We present PeerFlow, a load-balancing scheme for Tor that prevents an adversary from being directed a share of client traffic that is much larger than his share of the network capacity. PeerFlow uses a bandwidth-weighted voting process that resists manipulation by low-bandwidth adversaries, and it can use measurements from trusted relays for improved security. PeerFlow solves many of the additional challenges to creating a complete replacement for TorFlow, including a secure bootstrapping process for new relays and techniques to ensure user privacy in reported traffic statistics. We also prototype PeerFlow in the actual Tor software and use large-scale network simulations to show that its load-balancing maintains Tor's current performance.

5.2 Background and Related Work

Tor Tor [19] anonymizes a client TCP connection by randomly choosing three *relays* from its network, creating a multiply-encrypted *circuit* over those relays, and creating a *stream* through that circuit, which causes the endpoint to create an associated TCP connection to the destination. Streams can be multiplexed over a circuit.

Clients have a small, longstanding set of relays (*guards*) from which they select the first hop on their circuits. Tor clients use one guard rotated every 2–3 months. To become guards, relays must be old enough, provide at least 250KB/s of bandwidth, and have adequate uptime. Relays meeting these criteria receive the Guard flag, which makes them eligible to be selected as guards. Circuits also have a second (*middle*) hop and a third (*exit*) hop. Relays must allow connection to the client's desired destination port and IP to be selected as the exit. Most exits receive the Exit flag, given when a relay allows exit to the most useful ports. Currently relays must also receive the Fast flag (requires a bandwidth of 100KB/s) to be selected at all.

A system of *Directory Authorities* maintains flag and other relay information and publishes an hourly *consensus*, which clients use to select relays for circuits. Clients choose relays for a circuit position randomly proportional to the product of each relay's *consensus weight*, which is proportional to the relay's bandwidth for load balancing in a given position, and a *position weight*, which is based on the relay's flags and used to improve load-balancing across the different relay positions (namely, guard, middle, and exit). Relays also provide the Directory Authorities a self-determined *advertised bandwidth* to aid in setting their consensus weights.

Tor security requires that most of the network by consensus weight is not malicious. An adversary that controls much of the network will be selected often by clients. When selected as a guard, he can apply website fingerprinting [56,57] to identify the client's destination. When selected as guard and exit, he can deanonymize the connection using a first-last correlation attack [36]. And, of course, when selected for all three relays on a circuit, the connection can be trivially deanonymized.

Bandwidth Measurement TorFlow [2, 43] is a bandwidth-scanning tool that is currently used by Tor measuring authorities to directly measure the bandwidths of Tor relays and balance load among them. Past work has shown that TorFlow is vulnerable to multiple attacks [4, 51], which we will explore in more detail in Section 5.3. Our goal with PeerFlow is to use peers' direct observations of each other instead of centralized and authoritative bandwidth scanning in order to determine which relays can handle more traffic. EigenSpeed [46–48] is another scheme for accurate and secure peer-to-peer bandwidth estimation. Presented as a scheme to secure bandwidth evaluation in the Tor network, it is also proposed more generally for use in peer-to-peer distribution networks. EigenSpeed is the state of the art for this problem as far as we are aware, and it has been seriously considered for adoption

into Tor [1]. We show in Section 5.4 that EigenSpeed can be manipulated through several attacks and does not achieve its security goals.

Other Karame et al. [37] describe attacks on link capacity estimation techniques and suggest using trusted network hardware to secure these measurements. Suselbeck et al. [50] propose a system for estimating peer bandwidth in a P2P system. It includes both passive measurements and active traffic injection, as PeerFlow does, but assumes all peers are trustworthy, as PeerFlow does not. Haeberlen et al. propose the PeerReview system [27], which users cryptographic logs of node activity and witness audits of a node's actions to detect misbehavior in a distributed system. While these methods might prevent some kinds of misbehavior in PeerFlow, they are unable to solve major challenges, including (*i*) exposing falsely-claimed transfers between malicious relays, (*ii*) identifying trustworthy witnesses in a system where Sybil attacks are possible. Jansen et al. [34,35] describe how secure bandwidth measurements in Tor could be used to build a system to incentivize Tor relay operators by rewarding them for transferring traffic.

5.3 Attacks on TorFlow

TorFlow TorFlow [2,43] is a tool used by bandwidth-measuring Directory Authorities to directly measure the bandwidths of Tor relays. TorFlow works by constructing a series of *measurement circuits*, using them to perform test downloads, and then computing a weight for each relay based on the speeds of the test downloads.

TorFlow selects which relays to measure by dividing the list of all relays into *slices* of 50 relays of similar bandwidth (according to the most recent consensus). It measures each slice by constructing two-hop measurement circuits using only relays

[1] `https://trac.torproject.org/projects/tor/ticket/5464`

from that slice. When each circuit is built, TorFlow uses it to download a file from `torproject.org`. TorFlow continues building new circuits, choosing unmeasured relays from the current slice at random, to measure two relays at a time until every relay in the slice under examination has been measured several times. Then, for each relay, TorFlow takes the average bandwidth measured on circuits involving that relay, and stores this measurement to disk.

Every hour, TorFlow aggregates these measurements and produces a weight for each relay. A relay's weight is calculated by multiplying the relay's self-advertised bandwidth by the ratio between the measured bandwidth for that relay and the averaged measured bandwidth over the entire network. The Directory Authorities use these weights to produce the consensus weights.

Attacks TorFlow allows a number of attacks that make it easy to manipulate a relay's apparent bandwidth. First, as shown in [4], malicious relays can perform a *liar attack*, wherein they dishonestly report a higher bandwidth than they have available to increase their chances of being chosen during path selection while expending very few resources. TorFlow attempts to address this problem by adjusting self-reported bandwidths by a multiplier representing relative performance, but by continuing to use the reported bandwidth as a baseline, it remains vulnerable to the same attack. For example, a relay providing only 100 KB/s of bandwidth (the minimum required to obtain the Fast flag) could advertise a bandwidth so high that even after being adjusted down by TorFlow (as explained above), it will only be capped by 10 MB/s—the upper bound the Directory Authorities will assign to any relay. This attack is very effective, but gross exaggeration could be detected.

A more subtle attack takes advantage of TorFlow's two-hop measurement circuits, which are built with one of a small number of authorities on one end and a fixed URL

on the other. Because the IP addresses of these measurement nodes are known, relays can easily recognize measurement circuits and treat them differently from ordinary ones, a technique demonstrated by Thill [51]. In a *selective denial-of-service (DoS) attack*, a relay can provide service only to measurement circuits while dropping all others, thereby giving the authorities the impression of excess capacity at very low cost. An adversarial exit can further reduce resource consumption by spoofing short responses from the destination instead of downloading and serving real ones, since TorFlow does not verify certificates or check the correctness or length of downloads.

Results To demonstrate the efficacy of both the liar and selective DoS attacks, we developed a new TorFlow plug-in for the Shadow [33] discrete-event network simulator. The plug-in mimics the functionality of the python scripts [2] that are used to run TorFlow in the public Tor network. Using Shadow and our new plug-in, we constructed a private Tor network with 50 relays, 1000 clients, and 1 bandwidth authority that runs TorFlow. More details about Shadow, our TorFlow plug-in, and our Tor model can be found in Section 5.7.

We implemented both attacks outlined above in the Tor code base, and compiled the Shadow Tor plug-in with our modified Tor code. We configured one exit relay with a 1 gigabit/s access link and set its initial consensus weight to the same as the fastest relay in the network. Figure 5.1 shows the results of two simulations: one where our exit was acting honestly, and one where it was running both the liar and selective DoS attacks. We monitored the exit's bandwidth usage and its consensus weight fraction, the fraction of the total consensus weight that our exit achieved, over time.

The results shown in Figure 5.1 indicate that, using the attacks, the exit node was successfully able to *inflate* its consensus weight relative to other relays while at the

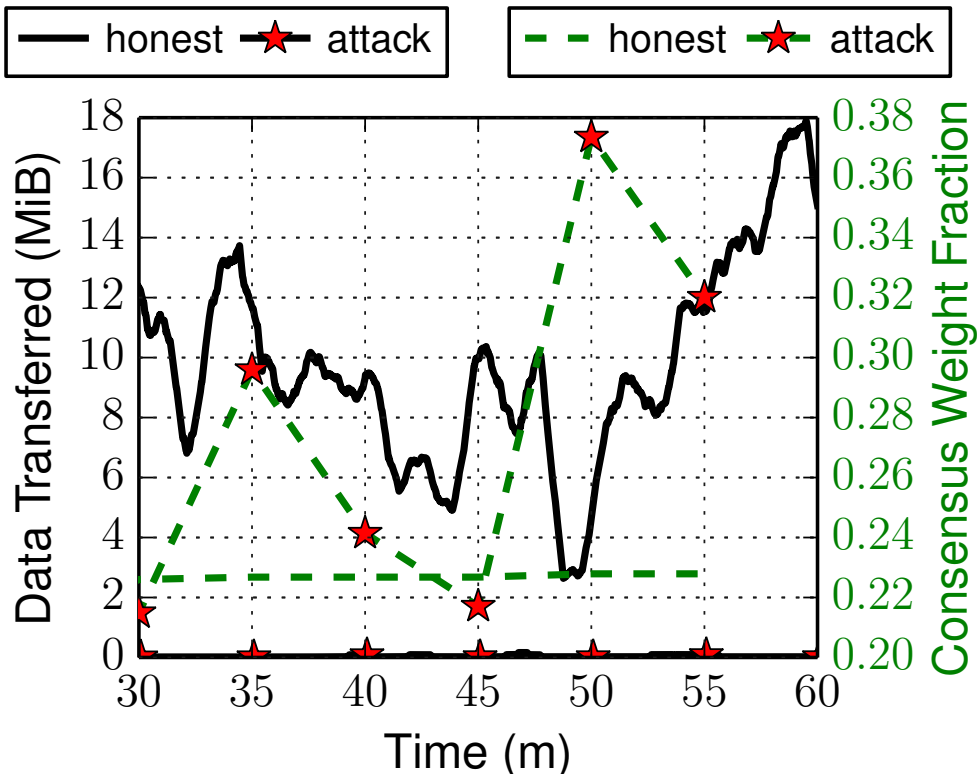


Figure 5.1: A relay can inflate its consensus weight at little cost by lying about its capacity and denying service to all but measurement circuits. Our experiments led to a bandwidth inflation factor of 489.476.

same time consuming *significantly less* bandwidth (only what was required for the measurement circuits). Our attack reduced the median bandwidth consumed by our exit from 9.395 MiB/s to 0.084 MiB/s, while the median consensus fraction obtained increased from 0.227 to 0.267; our attack enabled the exit to obtain more units of consensus weight fraction per bandwidth unit cost with a bandwidth inflation factor of 489.476. While we used only one relay for demonstration purposes, an adversary could easily use the same techniques with hundreds of relays to gain an even larger total fraction of the consensus weight.

We note that while TorFlow's current design makes these attacks easy to carry out, any bandwidth-scanner approach will need to solve the problem of relays detecting and giving biased service to measurement probes, as observed in [6].

5.4 Attacks on EigenSpeed

EigenSpeed EigenSpeed uses as consensus weights the eigenvector of a matrix derived from the observed bandwidth matrix T, where T_{ij} is the bandwidth observation of relay j by relay i. This process can be viewed as finding weights that are consistent with using themselves in a weighted average of the bandwidth observations. Tor's Directory Authorities are the recipients of each node's measurements of the others and are responsible for determining the weights. Snader's thesis [46] is the final and most complete description of EigenSpeed, and so we use it as the as the authoritative version.

The bandwidth observations measure the current bandwidth of a flow with a relay, rather than that relay's entire bandwidth. Each new bandwidth observation is combined with the current estimate using an exponentially-weighted moving average (EWMA). These weights are put into matrix T which the Directory Authorities use to produce $\overline{T}$ by first setting $\overline{T}_{ii} = 0$ to ignore self-measurements, next setting $\overline{T}_{ij} = \min(T_{ij}, T_{ji})$ to enforce symmetric measurements, and finally normalizing rows of $\overline{T}$ to sum to one. The EigenSpeed consensus weights are the left principal eigenvector v^* of $\overline{T}$, which is produced by iteratively computing $v^i = v^{i-1}\overline{T}$, where v^0 has a $1/t$ entry in the positions of a set of t *trusted relays* and a 0 entry elsewhere. Any relay j with sufficiently different measurements than the eigenvector, that is, with $\|v^* - \overline{T}_j\|_4 > L$ for $L = 10^{-5}$, is considered a liar and is removed and added to a set of *unevaluated* relays. Call this value the *liar metric*. Similarly, any relay j whose weight increased too fast during the first two iterations, that is, with $(v_j^2 - v_j^1)/v_j^1 > \Delta$ for $\Delta = 0.1$, is considered to be malicious and is removed and considered unevaluated. Call this value the *increase metric*. The unevaluated set also includes relays that are not in the largest component of the *measurement graph* (*e.g.* new relays), where an edge

(i, j) exists in the graph if $\overline{T}_{ij} > 0$. In EigenSpeed, unevaluated relays will each get $1/n$ of the total consensus weight, where n is the total number of relays.

Attacks An obvious vulnerability of EigenSpeed is that it selects each unevaluated relay with probability $1/n$. An adversary can flood the network with a large number of new relays contributing little or no bandwidth and thereby obtain a large total selection probability. Each new relay need only have a unique IP address. This could, for example, allow a botnet of just 20,000 computers to have a collective consensus weight of over 74% in a Tor network of its current size of about 7,000 honest relays. We therefore assume that unevaluated relays are selected with very low probability (*e.g.* if there are u unevaluated relays, then the probability of selecting a given one is $0.01/u$). This limits the effect of a Sybil attack to just taking over the unevaluated set, but it also means that a relay that is unevaluated can be essentially shut out of the Tor network.

We show how the two mechanisms that EigenSpeed uses to detect malicious nodes can be used to frame honest relays and have them put into the unevaluated set. The liar threshold $L = 10^{-5}$ and increase threshold $\Delta = 0.1$ are designed for an attack in which a clique of malicious relays report high bandwidth with each other and low bandwidth with others. However, modifications to this attack can confuse the trusted relays (which otherwise help distinguish between the honest and dishonest relays) by making the liar or increase metrics for honest framed relays appear large. This will imply that either (i) the adversary can get the honest non-trusted relays effectively kicked out of the network by moving them to the unevaluated set or (ii) L and Δ are large enough that the adversary can greatly inflate the inferred bandwidth of his relays.

Analysis We demonstrate framing attacks using the Tor 2015-04-31 23:00 network consensus [11], which contains 5589 relays with positive selection probability (over 1000 relay have zero selection probability). EigenSpeed measurements are correlated with the per-stream bandwidth at a given relay rather than the total bandwidth at that relay (see [46] Section 3.3.1). Thus we consider the ideal load-balanced case, in which all streams have the same bandwidth. We note that our attacks are even more effective when the network is not load-balanced, in which case there is disagreement among the relays' observations, forcing the thresholds L and Δ to be large. For example, suppose that the relay observations are the total relay bandwidths, which we take to be the "observed bandwidths" in the relays' descriptors [52]. Then, with 10% of relays trusted (starting with the largest by observed bandwidth) and no malicious relays, honest relays have a liar metric as large as 0.00165, much larger than the suggested threshold of $L = 10^{-5}$, and honest relays have an increase metric as large as 0.86, much larger than the suggested $\Delta = 0.1$. We thus consider all our attacks in the load-balanced setting, in which case these metrics are much smaller and well below the suggested thresholds. In all of our experiments, we follow Snader [46] and stop the iterative eigenvector calculation when the change in vector norm is less than 10^{-10}.

The *increase framing attack* causes the increase metric of a targeted set of honest relays to be large. In this attack, a set of malicious relays obtains the average bandwidth measurement with all trusted relays and with a subset of "framed" honest, non-trusted relays. The malicious relays also falsely claim that bandwidth measurement among themselves (*i.e.* no data is actually sent among them). All other measurements with malicious relays are zero. All measurements among the honest relays are the same load-balanced flow rate (say, 1). Note that each malicious relay can easily obtain any bandwidth measurement with any honest relay, as long as the

# trusted relays (% of honest)	# adv relays	# framed relays	Adv bw %
280 (5%)	447	1118	1.92
559 (10%)	558	1118	2.83
1118 (20%)	558	559	2.83
1677 (30%)	558	112	3.00

Table 5.1: Cases with minimum bandwidth in which all framed relays had increase metrics above 0.2.

measurement doesn't exceed the relay's true bandwidth, by creating spurious connections to the honest relay with the necessary amounts of traffic. The malicious relay may furthermore drop all connections from honest clients for measurements of zero.

We consider adding relays to the Tor network in order to frame a subset of honest relays by causing their increase metric to reach above 0.2. Not only is this amount is greater than the $\Delta = 0.1$ recommended, it will be enough to allow significant bandwidth inflation and thereby demonstrate that no setting for Δ can provide good protection. For various numbers of trusted relays, we searched for a minimum adversarial bandwidth needed to frame at least 2% of the relays. We count the bandwidth of a relay as the sum of the measurements obtained with each other relay, ignoring the false measurements among malicious relays. Table 5.1 presents our results. It shows that with at most 3% of the total bandwidth, the adversary can frame between 112 (2%) and 1118 (20%) of the honest relays, with the number decreasing as the number of honest relays that are trusted increases from 280 (5%) to 1677 (30%). The number of relays that the adversary must add is rather small, from 447 to 558. Moreover, the attack can easily be repeated (with a different set of malicious relays) in order to move even more honest relays into the unevaluated set. As we noted, the selection probability for relays in the unevaluated set must be quite low. Therefore, they will be rarely observed by other relays, and so they must either wait many measurements periods to be evaluated or will have very low inferred bandwidths. In

# trusted relays (% of honest)	# adv relays	False bw factor	Adv bw %	Adv weight %
280 (5%)	4191	100	3.49	98.2
559 (10%)	2235	100	3.70	93.9
1118 (20%)	1117	100	3.70	79.5
1677 (30%)	1397	15	6.52	48.5

Table 5.2: Cases with maximum weight in which all malicious relays had increase metrics below 0.2 and liar metrics below the honest non-trusted relays.

addition, if the adversary performs another Sybil attack on the unevaluated pool, which requires IP addresses but no bandwidth, then relays in the unevaluated pool are effectively removed entirely. Even worse, the relay bandwidths are highly skewed, and so even for the smallest number of frame relays in Table 5.1 (112), the adversary quickly cause most of the network capacity to be unused. For example, in the consensus used in our experiments, 50% of the total observed bandwidth is provided by the largest 464 relays and 75% by the largest 1172.

We next show how to confuse the trusted relays in order to keep the liar metrics of the malicious relays low while inflating their weight. In these experiments, the malicious relays create bandwidth measurements with the trusted relays that are the average bandwidth of the honest non-trusted relays, the malicious relays create measurements of zero with the honest non-trusted relays, and the malicious relays report a large false bandwidth among themselves. All honest relays again have the same bandwidth measurement among themselves.

Thus the trusted relays make the same measurements with the other honest relays as with the malicious relays, but the measurements of the other honest relays are in disagreement with those of the malicious relays. This makes the attack use less bandwidth while keeping the liar metrics of malicious relays close to those of honest non-trusted relays. Table 5.2 lists the scenarios in which the final adversary weight was maximized among those we tested subject to (*i*) preferring adversary bandwidth of less than 4% when possible, (*ii*) producing increase metrics for mali-

cious relays of no larger than 0.2, and (*iii*) producing liar metrics of malicious relays less than those of honest non-trusted relays. We can assume that $\Delta > 0.2$, or the previously-described framing attack would be possible. Therefore, no malicious relays are unevaluated due to an increase metric above Δ. Furthermore, no setting of the liar threshold L is able discriminate between malicious and honest relays. Either it would allow the large adversary weight inflations shown in Table 5.2 or it would put all honest non-trusted relays in the unevaluated set. We can see that by reporting a false bandwidth of 15-100 times the load-balanced rate reported by all honest relays, malicious relays can potentially obtain weight inflated 7.4-28.1 times.

5.5 PeerFlow

PeerFlow uses two relay-measurement techniques: (*i*) each relay reports on the number of bytes it sent to or received from other relays, and (*ii*) each relay reports its available but unused bandwidth. Measurements from the first technique are combined to estimate the total bytes transferred after trimming a weight fraction λ of the largest and smallest values, and thus an adversary without λ of the network capacity can't manipulate the outcome. Measurements from trusted relays (if available) are used to ensure that the estimates of bytes transferred aren't unreasonably high or low. The second measurement technique allows the network to discover any unused bandwidth. It is vulnerable to lying by an adversary, though, and therefore PeerFlow will only consider increasing the consensus weight for a relay after consulting the secure outputs of the first measurement technique and verifying that the relay carried the expected amount of traffic. Further methods are used to increase the privacy, accuracy, and speed of collecting these measurements and to securely bootstrap new relays into the system. We now explain each of the PeerFlow components in detail.

Name	Description
β_p^{SR}	*bytes* measured by S to and from R in position p
$\rho_{(p)}^{(S)R}$	total traffic *relayed* by R (inferred by S, in position p)
$\eta^{(S)R}$	*estimated* traffic relayed by R (with S)
κ^R	*capacity* of relay R computed by Directory Authorities
σ^R	*self*-estimated capacity of relay R
ω^R	consensus *weight* of router R
v_p^R	*voting* weight of relay R in position p
t^R	measurement *time* of relay R
λ	fraction of measuring relay inferences to trim
τ	trusted relay weight fraction in each position
ϵ_{dec}	max fraction a relay's weight can decrease
μ	weight of measuring relays: 0.75
ϵ_{inc}	max fraction a relay's weight can increase: 0.25

Table 5.3: Key variables (top), parameters (bottom) in PeerFlow

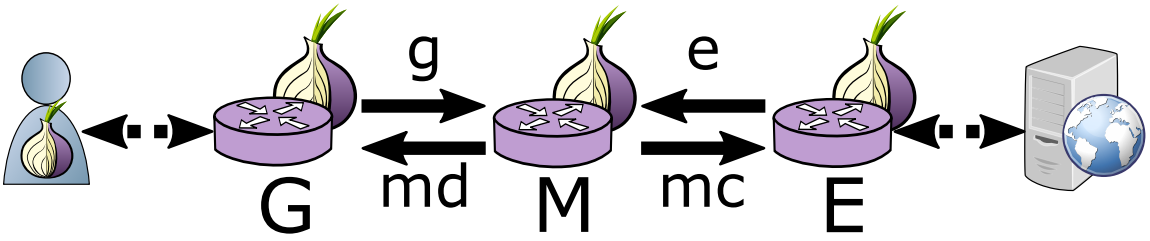


Figure 5.2: Measuring positions in a circuit: relay G in position g measures relay M, relay M in position mc measures relay E, relay M in position md measures relay G, and relay E in position e measures relay M. A dashed arrow indicates hosts between which traffic is not measured.

For convenience, the key variables and parameters are listed in Table 5.3.

5.5.1 Measuring total traffic of a relay

For each circuit position, a subset of relays keeps track of the amount of traffic it sends to and receives from all of the other public relays while in that position. Let $\mathcal{M}_g$, $\mathcal{M}_m$, and $\mathcal{M}_e$ be the set of *measuring guards*, *measuring middles*, and *measuring exits*, respectively. $\mathcal{M}_p$ is defined to be the set containing each relay R with positive *voting weight* v_p^R in position p, that is, $\mathcal{M}_p = \{R | v_p^R > 0\}$. Voting weights approximate the relays' relative capacity in each position, with the smallest relays excluded to speed up measurement (see Section 5.5.6).

Each measuring relay $r \in \mathcal{M}_p$ counts the number of bytes exchanged with each

other relay while r is in position p. Bytes are counted at the application layer, and both sent and received bytes are included in the count. A measuring relay detects itself in the guard position if the circuit-creation messages are not sent by a public Tor relay, it detects itself in the exit position if the circuit is not extended past the measuring relay, and otherwise it assumes the middle position. A measuring middle further divides its observations into *client-side* and *destination-side* measurements according to whether the measuring relay extended the circuit to the measured relay during circuit creation or vice versa, respectively. A measuring relay R_0 counts traffic sent to and received from each relay R_1 for a measurement period of length t_{R_1} that depends on the bandwidth of R_1 (see Section 5.5.4). Let $\beta_p^{R_0R_1}$ be the number of bytes that measuring relay R_0 sent to or received from R_1 during the measurement period while R_0 was in *directional position* p. The possible directional positions are guard, client-side middle, destination-side middle, and exit, denoted g, mc, md, and e, respectively (see Figure 5.2).

At the end of the measurement period for relay R_1, measuring relay R_0 will process the traffic count for each position that it measures for and send it to the Directory Authorities. The first step in processing count $\beta_p^{R_0R_1}$ protects the privacy of individual traffic flows by adding noise. The noise is a random value selected from the Laplace distribution $\mathsf{Lap}(b)$, which has mean 0 and variance $2b^2$, where b is set to provide differential privacy for a certain amount of traffic on the link (see Section 5.5.3). Let $N_p^{R_0R_1} \sim \mathsf{Lap}(b)$ be the noise value, and let $\tilde{\beta}_p^{R_0R_1} = \beta_p^{R_0R_1} + N_p^{R_0R_1}$.

The second processing step is to infer the total amount of traffic to or from R_1 seen in directional position p. This is accomplished by adjusting the amount R_0 sees by the probability of making that observation. Let $q_g^{R_0}$ be the probability of selecting R_0 as a guard, as determined from the consensuses of the measurement

period. Let $q_m^{R_0}$, $q_{mc}^{R_0}$, and $q_{md}^{R_0}$ all be set to the probability of selecting R_0 as a middle. Let $q_e^{R_0}$ be the probability that R_0 is chosen as an exit (using the presence of the Exit flag as an approximate way to identify possible exits). Then let the estimate for the total traffic relayed by R_1 and seen in directional position $p \in \{g, mc, md, e\}$ be $\rho_p^{R_0 R_1} = \tilde{\beta}_p^{R_0 R_1} / q_p^{R_0}$.

At the end of the measurement period for R_1, the Directory Authorities will receive the ρ statistics about R_1 from the measuring relays. The Directory Authorities remove the largest and smallest fraction λ of the ρ values for each directional position $p \in \{g, mc, md, e\}$ and aggregate the remaining values to obtain an estimate $\bar{\rho}_p^{R_1}$. This process removes the influence of outliers, including especially any falsified reports from malicious relays. When trimming the fraction λ, a voting weight v_p^R proportional to R's capacity is used for each estimate $\rho_p^{RR_1}$, and so an adversary that provides little bandwidth to Tor has minimal effect on the measurement outcomes. Let i_j be the index of the relay R_{i_j} with the jth largest value $\rho_p^{R_{i_j} R_1}$, let j_1 be the largest value such that $\sum_{j<j_1} v_p^{R_{i_j}} < \lambda$, and let j_2 be the smallest value such that $\sum_{j>j_2} v_p^{R_{i_j}} < \lambda$. The trimmed ρ statistics are combined by adding their noisy byte values and dividing by the total selection probability of the untrimmed relays: $\bar{\rho}_p^{R_1} = \sum_{j_1 \leqslant j \leqslant j_2} \rho_p^{R_{i_j} R_1} q_p^{R_{i_j}} / \sum_{j_1 \leqslant j \leqslant j_2} q_p^{R_{i_j}}$.

When there are no trusted relays, we set $\lambda = 0.256$ to maximize the size of the adversary that is prevented from arbirtrarily increasing his weight. PeerFlow can defend against (see Section 5.6). Note that using the median (*i.e.* $\lambda = 0.5$) does not provide optimal security in this case. When there are trusted relays, we set λ to maximize the size of the adversary such that PeerFlow's method of using μ of the network for measurement results in a smaller bound on an adversary's capacity increase compared to simply using measurements from the smaller set of trusted relays. For $\tau = 0.05$, 0.1, 0.2, and 0.3, the respective values of λ are 0.34, 0.348,

0.497, and 0.498.

Given the aggregate values $\bar{\rho}_p^{R_1}$, the Directory Authorities calculate two estimates for the total number of bytes relayed by R_1. The first is the sum of the client-side estimates, $\rho_c^{R_1} = \bar{\rho}_g^{R_1} + \bar{\rho}_{mc}^{R_1}$, and the second is the sum of the destination-side estimates, $\rho_d^{R_1} = \bar{\rho}_{md}^{R_1} + \bar{\rho}_e^{R_1}$. If R_1 can act as a guard, then client-side observations for it will be missing for any circuits on which it is a guard, and similarly for destination-side observations if R_1 can act as an exit. On the other hand, when a relay acts as a middle it is observed both on the client and destination side but should get credit for that traffic only once. Therefore, the Directory Authorities use $\rho_{\max}^{R_1} = \max(\rho_c^{R_1}, \rho_d^{R_1})$ as an estimate for the total amount of traffic relayed by R_1. In order to avoid excluding client-side or destination-side observations made on separate circuits, PeerFlow requires that during a measurement period a Tor relay will not operate both as a guard and as an exit (Tor effectively already enforces this currently via its bandwidth weights, as exit bandwidth is relatively scarce and is thus reserved for exiting).

PeerFlow takes advantage of measurements from any trusted relays by using them to limit the range in which a relay's traffic will be inferred. PeerFlow uses trusted relays in this way instead of simply using only the trusted measurements because doing so provides better security and accuracy when relatively little of the network is trusted. Because many of the highest-bandwidth Tor relays are managed by organizations closely aligned with the Tor Project, and (as of 15 May 2015) the top 60 relays constitute over 20% of the total weight, it seems reasonable to imagine a set of trusted relays that carry 15-25% of Tor traffic.

Let $\mathcal{T}_p$ be the set of trusted relays in position p, and let τ be the minimum fraction of relay capacity that they are assumed to provide in each of the guard, middle, and exit positions. At the end of a measurement period for R_1, the Directory Authorities

simply combine the measurements from the trusted relays of bytes exchanged with R_1 to determine the following trusted estimate for the total bytes relayed by R_1 and observed in position p: $\hat{\rho}_p^{R_1} = \sum_{R \in \mathcal{T}_p} \rho_p^{RR_1} q_p^R / \sum_{R \in \mathcal{T}_p} q_p^R$. PeerFlow then combines these positional trusted estimates as it does with the analogous estimates from all relay measurements to produce $\hat{\rho}_c^{R_1} = \hat{\rho}_g^{R_1} + \hat{\rho}_{mc}^{R_1}$, $\hat{\rho}_d^{R_1} = \hat{\rho}_{md}^{R_1} + \hat{\rho}_e^{R_1}$, and $\hat{\rho}_{\max}^{R_1} = \max(\hat{\rho}_c^{R_1}, \hat{\rho}_d^{R_1})$.

The trusted estimate $\hat{\rho}_{\max}^{R_1}$ is used to adjust the all-relay estimate $\rho_{\max}^{R_1}$ by enforcing a ceiling and floor on its value. The ceiling is simply $\hat{\rho}_{\text{ceil}}^{R_1} = \hat{\rho}_{\max}^{R_1}$, which means that the inferred traffic relayed by R_1 will be limited by the number of bytes it can exchange with trusted relays. The floor is $\hat{\rho}_{\text{floor}}^{R_1} = \hat{\rho}_{\max}^{R_1} \tau / (\mu(1-\lambda))$, which both limits the amount that the adversary can use falsely low measurements from its relays to reduce honest relays' inferred traffic relayed and ensures that the adversary gains no advantage in targeting its bandwidth on the trusted relays instead of the top $1-\lambda$ fraction of measuring relays in a given position. The final estimate for the traffic relayed by R_1 is thus $\rho^{R_1} = \max(\min(\hat{\rho}_{\text{ceil}}^{R_1}, \rho_{\max}^{R_1}), \hat{\rho}_{\text{floor}}^{R_1})$. If there are no trusted relays, then $\rho^{R_1} = \rho_{\max}^{R_1}$.

5.5.2 Measuring available bandwidth

Each relay R monitors its network activity during a measurement period and estimates its total capacity for relaying traffic. That is, R estimates the maximum rate at which it could have relayed traffic during the measurement period if Tor clients had asked it to. This could be measured in the same way that Tor relays currently determine their self-advertised bandwidths. R sends this self-measured value σ^R to the Directory Authorities at the end of the measurement period.

5.5.3 Preserving link privacy with noise

The traffic statistics that measuring relays report to Directory Authorities reveal how traffic flows through the Tor network. Traffic statistics *per relay* are already collected and reported by Tor [11], but the PeerFlow statistics describe traffic between each *pair* of relays. These statistics should not assumed to be kept secret by the Directory Authorities, because some Directory Authorities may be compromised and because it allows auditing of PeerFlow to identify errors and relay misbehavior.

The risk of releasing the PeerFlow measurements is that they could be used to identify the routes through the Tor network taken by a target set of connections. For example, a malicious destination might target an incoming connection and use its observation of the exit node and traffic volume to identify as the middle node that relay measured by PeerFlow to have sent the same amount of traffic as the target connection. The guard relay could then be identified similarly. Although there exist other ways of identifying the relays used on a connection (*e.g.* the congestion attack [22], latency attack [28], and predecessor attack [59]), this information should still have some protection.

Therefore, measuring relays in PeerFlow add a random value to the observed byte totals. The goal of this added noise is to limit the certainty with which an adversary can conclude that a given client stream was carried between a given pair of relays. We accomplish this by choosing the noise value randomly from the Laplace distribution with mean 0, which has the probability density function $\mathsf{Lap}(x; b) = e^{-|x|/b}/(2b)$. Adding noise according to the Laplace distribution provides *differential privacy* [20], where the privacy notion applies to a given amount of traffic. We set the Laplace parameter to $b = \tau_n/\epsilon_n$, where τ_n is the amount of traffic for which the output distribution has bounded change (we use $\tau_n = 1$ MiB), and $\epsilon_n > 0$ is the bound on

that change (we use $\epsilon_n = 0.1$). Section 5.9 describes a cryptographic measurement-aggregation scheme that further limits the amount of traffic data revealed.

5.5.4 Measurement periods

The length of a measurement period is determined for each relay individually and is updated at the end of the each measurement period. We would like this length to be low in order to enable quick response to changes in relay bandwidth and load. The speed of measurement is limited by how quickly a relay can exchange an amount of client traffic with each measuring relay such that the added noise is relatively small.

Let $t_0^{R_1}$ be the length of a just-completed measurement period for relay R_1. The Directory Authorities update $t_0^{R_1}$ to t^{R_1} by using the recent measurements $\rho_{p0}^{R_0 R_1}$ to estimate the traffic rates with each measuring relay R_0 in each position p. t^{R_1} is set to a large enough value that with probability q_e for at least $1-2\lambda$ of the measuring relays by voting weight the amount of noise added is less than a fraction ϵ_e of the traffic exchanged with R_1 during the measurement period (we use $q_e = 0.1$ and $\epsilon_e = 0.1$). $1 - 2\lambda$ of the voting weight ensures accurate estimates remain after trimming the largest and smallest fraction λ of the ρ statistics. In addition, t^{R_1} is set large enough that $0.4\rho^{R_1} t^{R_1} \epsilon_{\text{dec}}(1-2\lambda)\mu\epsilon_n/(\tau_n t_0^{R_1}) \geqslant 2\max(|\mathcal{M}_g| + |\mathcal{M}_m|, |\mathcal{M}_e| + |\mathcal{M}_m|)$. This is for security purposes and will ensure that the added noise is small relative to the expected traffic of any adversarial relays.

5.5.5 Load balancing using measurements

The Directory Authorities use the aggregate peer-measurement ρ^R and self-measurement σ^R to produce the consensus weight ω^R for relay R. Tor clients select relay R for a given position in a new circuit with probability proportional to

ω^R (approximately, see Section 5.2). Thus, in order to balance the load across the available relays, the Directory Authorities attempt to determine consensus weights that are proportional to the bandwidth of each relay. They must do this even as relay capacities and network traffic change, and they must do it in a way that is secure against manipulation. They will accomplish this by computing and comparing the amounts of traffic a relay was expected to transfer, did transfer, and claims it can transfer.

Let η^R be the amount of traffic that R is expected to have relayed in the current measurement period. To determine η^R, let $\mathcal{P}^R$ be the set of relays (including R) that could be chosen for the same positions as R in their most-recently completed measurement period based on the Guard, Exit, and Fast consensus flags, where for this purpose a relay is considered to possess each of these flags if they exist for the majority of consensuses during the measurement period. Then let $\omega_0^{R'}$ be the weight used by $R' \in \mathcal{P}^R$ during its most-recently completed measurement period, let $t_0^{R'}$ be the length of that period, and let $\rho_0^{R'}$ be the inferred number of bytes relayed during that period. Finally, set η^R to be the voting-weight median of the set $\{\omega_0^R t_0^R \rho_0^{R'}/(\omega_0^{R'} t_0^{R'})\}_{R' \in \mathcal{P}^R}$.

Let κ^R be the current estimate for the *capacity* of R, that is, the rate of traffic-relaying that R has recently sustained. κ^R is set initially during the bootstrapping process. Let S^R be the measurement state of R. After the bootstrapping period has finished, S^R will only take values NORMAL and PROBATION (its use during bootstrapping is described in Section 5.5.7).

For S^R = NORMAL, if R is non-trusted, Figure 5.3 describes how the Directory Authorities update the consensus weight ω^R, the relay state S^R, and the capacity estimate κ^R. Observe that κ^R is never set to be larger than the relay's self-estimate σ^R and that κ^R is increased to the observed rate ρ^R/t^R if κ^R is less than it. κ^R will

1: $\kappa'^R = \min(\max(\rho^R/t^R, \kappa^R), \sigma^R)$
2: $\omega^R = \min(\max((1+\epsilon_{\text{inc}})\kappa^R, \kappa'^R), \sigma^R)$
3: $\kappa^R = \kappa'^R$
4: **if** $\rho^R < (1-\epsilon_{\text{dec}})\min(\kappa^R t^R, \eta^R) \wedge \rho^R/t^R < \sigma^R$ **then**
5: $S^R =$PROBATION
6: $\omega^R = \kappa^R$
7: **end if**

Figure 5.3: Non-trusted relay weight algorithm if $S^R =$ NORMAL

1: $\kappa^R = \min(\rho^R/t^R, \sigma^R)$
2: $S^R =$NORMAL
3: $\omega^R = \min((1+\epsilon_{\text{inc}})\kappa^R, \sigma^R)$

Figure 5.4: Non-trusted relay weight algorithm if $S^R =$ PROBATION

only be decreased if R transfers a certain fraction ϵ_{dec} less than both the expected amount η^R and R's estimated limit $\kappa^R t^R$. This fraction is set to $\epsilon_{\text{dec}} = 1-\tau/(\mu(1-\lambda))$ to protect honest relays from being forced into PROBATION by an adversary. Setting ϵ_{dec} this way protects an honest R because R will send the expected amount to the trusted relays, and so $(1-\epsilon_{\text{dec}})\eta^R = \hat{\rho}^R_{\text{floor}} \leqslant \rho^R$. Without trusted relays, ϵ_{dec} should be set to a small value to allow some natural variation in traffic amounts without allowing too much underperformance without penalty (*e.g.* $\epsilon_{\text{dec}} = 0.25$). If ρ^R falls below the required threshold but R indicates with σ^R that it believes it has a higher capacity than the amount measured, then R maintains its capacity but enters the PROBATION state. When the relay stays in the normal state, the final weight produced ω^R is only increased either to a newly-demonstrated capacity κ^R or, if R believes it has additional unused capacity, to a fraction ϵ_{inc} over the old capacity (we use $\epsilon_{\text{inc}} = 0.25$).

Probation allows a relay to avoid a weight decrease due to a random or malicious

lack of client traffic. When non-trusted relay R enters the probation state, it attempts to prove over the next measurement period that it is capable of transferring at a rate that is the minimum of its current capacity κ^R and its self-assessed capacity σ^R. To do so, R monitors the number of bytes that it transfers to other relays, and it sends and receives dummy traffic as needed to transfer $\min(\kappa^R, \sigma^R)t^R$ bytes total after t^R time. The dummy traffic is openly exchanged pairwise with the measuring relays, it includes traffic sent in both directions, and each measuring relay R' takes the minimum of dummy bytes sent and received and adds it to both $\rho_{mc}^{R'R}$ and $\rho_{md}^{R'R}$. As shown in Figure 5.4, at the end of the measurement period the relay leaves probation status and is assigned the capacity that is measured ρ^R/t^R, unless that exceeds its updated self-assessment.

Trusted relays also update their weights as shown in Figures 5.3 and 5.4, with the additional requirement that trusted relays maintain τ fraction of the weight (*i.e.* selection probability) in each position. After the update of ω^R for *any* relay R, an inflation factor is computed that is multiplied by the trusted-relay weights when computing selection probabilities q_p^R for $R \in \mathcal{T}$. The inflation factor is taken to be minimum value greater than one such that $\sum_{R \in \mathcal{T}} q_p^R \geqslant \tau$ for $p \in \{g, m, e\}$.

5.5.6 Updating voting weights

For accuracy and security, the voting weight of a relay should reflect the amount of bandwidth it has provided in a given position. Giving relays with high contributed bandwidth high voting weights improves accuracy because those relays have the most observations about other relays' activity. It improves security because it requires the use of costly bandwidth in order for an adversary's votes to affect relay weights by much, and because it requires that an adversary exchange a large amount of traffic with a large fraction of the network in order to obtain large weights for its relays.

Thus PeerFlow bases voting weights on previous estimates for the amount of relayed traffic. However, voting weights are updated more slowly than consensus weights in order to increase the upfront bandwidth cost of obtaining influence over consensus weights. In addition, a fraction of the smallest relays is excluded from voting to speed up measurement.

We need to maintain that the measuring relays constitute a significant fraction of the network. However, we would also like to update voting weights infrequently to force an adversary to relay traffic for a while before increasing his voting weight. To satisfy the former concern, we select a fraction μ of the network capacity in each position to receive a positive voting weight (we use $\mu = 0.75$). To satisfy the latter, we only update the voting weights when the cumulative selection probability of any $1 - 2\lambda$ of the voting weight in some position falls below $(1 - 2\lambda)\mu$. This will ensure that an small adversary (*i.e.* one with voting weight $\alpha < \lambda$) can't inflate his weight by targeting the $1 - 2\lambda$ fraction of measuring relays that have had their selection probability reduced the most.

The Directory Authorities initiate any update of the voting weights after determining the next consensus weights. To perform an update, the Directory Authority first estimates the amount the relays have each relayed in a given position. Let ρ_v^R be the total estimated traffic relayed by R since the last voting-weight update. That is, ρ_v^R is the sum over the measurement periods that completed for R since the last voting-weight update of the estimates ρ^R obtained in each period, as described in Section 5.5.1. Using the weights of the next consensus, for each position $p \in \{g, m, e\}$ and relay R, the Directory Authority multiplies the position weight for p by ρ_v^R to obtain a weight $\rho_{v,p}^R$. Let $i_{j,p}$ be the index of the relay with the jth largest value $\rho_{v,p}^R$. Then, for position p, the Directory Authority gives a positive voting weight to the relays that have relayed the most traffic and constitute a fraction μ of the

network capacity. Let κ_p^R be the current estimated capacity κ^R of relay R multiplied by the position weight for R and p. Let j_p^* be the number of relays needed to reach a fraction μ of $\sum_R \kappa_p^R$. Relay $R_{i_{j,p}}$, $j \leqslant j_p^*$, is assigned a voting weight for position p of $v_p^{R_{i_{j,p}}} = \kappa_p^{R_{i_{j,p}}} / \sum_{k \leqslant j_p^*} \kappa_p^{R_{i_{k,p}}}$. Relay $R_{i_{j,p}}$, $j > j_p^*$, is assigned a voting weight for position p of $v_p^{R_{i_{j,p}}} = 0$.

5.5.7 Bootstrapping new relays

PeerFlow boostraps new relays into the system using the following staged process:

Initialization A new relay that Tor would currently just add to the next consensus has its measurement state S^R initialized to UNKNOWN.

Unknown In any consensus period a set of *Bandwidth Authorities*, such as those currently used by TorFlow, estimate the capacity of a relay R with S^R =UNKNOWN by downloading a set of test files of increasing size through a one-hop circuit consisting of R. The test file is obtained from a Bandwidth Authority itself, and the Bandwidth Authority should have enough capacity to measure a reasonable lower-bound on capacity for the largest relays. To reduce the ability of the adversary to slow down the testing process, Bandwidth Authorities test relays in the order that they joined, and they only allow the download to take as long as it would take a relay with the minimum amount of bandwidth needed for the Fast flag (currently 100KB/s [52]).

Regardless of whether all downloads finished, at the end of the tests for relay R, the Bandwidth Authority estimates a sustainable rate for the relay κ^R as the minimum of the observed rate (*i.e.* the test bytes transferred over the time needed to transfer them) and the initial self-measured capacity σ^R. The weight of R is

initialized to $\omega^R = \kappa^R$. The measurement time t^R is set as described in Section 5.5.4, except the traffic amounts ρ_p^R in each measurement position p (which don't exist initially) are estimated by using those amounts $\rho_p^{R'}$ from the other relays R' with the same flags as R after their most-recently completed measurement period. To estimate ρ_p^R from these, each $\rho_p^{R'}$ is normalized to $\omega^R \rho_p^{R'}/(t^{R'}\omega^{R'})$, and ρ_p^R is set to the vote-weighted median of these values. Finally, the state of R is updated to $S^R =$ ESTIMATED.

Note that this stage is for performance reasons only. The amount of data downloaded during the tests is not high and is not intended as a security barrier. In addition, the measured relay is aware that the requested downloads are measurement tests performed by Bandwidth Authorities.

Estimated Relay R with $S^R =$ESTIMATED starts being selected only for the middle position when doing so won't cause the selection probability of estimated relays to exceed some small amount (*e.g.* 5%). The middle position cannot observe the client or destination directly, and so PeerFlow allows relays to occupy the middle position based only on an insecure capacity estimate. The limit on the probability of estimated relays limits the extent of observations they can make. Note that estimated relays are not considered when determining the measuring relays, which, among other things, prevents an adversary from triggering a voting-weight update by flooding new relays.

Measuring relays measure R just as they do for relays in the NORMAL state. After t^R time, an estimate ρ^R is produced for the amount relayed by R using the same trimmed vote as for relays in the NORMAL state. Then the capacity estimate is updated to $\kappa^R = \min(\rho^R/t^R, \sigma^R)$, and the consensus weight is set to $\omega^R = \kappa^R$. Finally, the relay's state is updated to $S^R =$ NORMAL.

5.6 Security analysis

The main security goal of PeerFlow is to ensure that an adversary that relays a fraction ϕ of Tor's traffic only obtains a total relative consensus weight of $\gamma\phi$, where γ is a small advantage factor. We first examine consensus weights in a given voting-weight period (voting-weight periods are described in Section 5.5.6). We show bounds on γ, which, in addition to the limits imposed by trusted relays, show that if the adversary is small (*i.e.*, has a maximum voting-weight fraction of $\alpha < \lambda$), the advantage is bounded even without any trusted relays. We then examine the consensus weights across voting-weight periods and show that PeerFlow provides bounded advantage there as well.

5.6.1 Weights in a single voting-weight period

Let α_p, $p \in \{g, m, e\}$, be voting weight fraction that the adversary's relays cumulatively possess in position p, and let $\alpha = \max_p \alpha_p$. Let A be an adversarial relay. Let β^A be the total number of bytes sent or received by A during its measurement period. We use the variables in Table 5.3 to refer to those values in the current consensus, and we denote by x_0 the value of variable x in the last consensus (*e.g.* κ_0 indicates the capacity inferred at the end of the last consensus).

We focus on $\alpha < \lambda$ because initially $\alpha = 0$, small adversaries will maintain small α (as shown in Section 5.6.2), and with $\alpha > \lambda$ the adversary's inferred capacity is bounded only by the trivial trusted ceiling. Lemma 1 shows that the number of bytes that PeerFlow infers were transferred by A (*i.e.* ρ^A) is expected to be at most a constant multiple of β^A plus a small fraction of the bytes of the bytes expected if A continues at its last rate of ρ_0^A/τ_0^A. For example, the multiple of β^A is 1.8 for $\alpha = 0$ and $\lambda = 0.256$. The additional term is due to the added noise, and at $\epsilon_{\text{dec}} = 0.25$ it

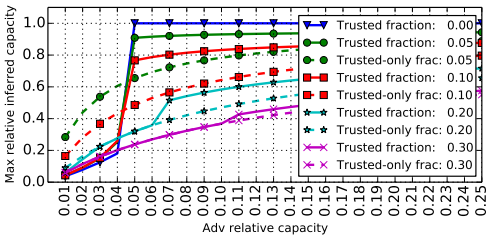


Figure 5.5: Max adversarial relative inferred capacity across voting periods using trusted relays and using trusted relays *only*

will be 0.1 times β^A if A continues to send at last inferred rate.

Lemma 1. *If $\alpha < \lambda$, then*

$$\mathbb{E}[\rho^A] \leqslant \frac{\beta^A}{(1-\lambda-\alpha)\mu} + \frac{0.4\epsilon_{dec}\rho_0^A t^A}{t_0^A}.$$

Proof. See Section 5.10. □

Lemma 1 limits the amount an adversary can increase his relays' inferred bytes transferred. His total consensus weight is relative to the honest relays, however, and it may occur that many of these relays transfer most of their traffic with a small number of measuring relays, whose values get trimmed. However, it is unlikely that traffic from honest clients is skewed much in this way. Let $\gamma' \leqslant 1$ be the smallest factor by which some set of honest measuring relays with voting weight $1 - 2\lambda$ underestimates an honest relay. Let $\mathcal{A}$ be the set of indices of the adversary's relays. Lemma 4 states that the inferred bytes of each honest relay is at most γ' of the correct value.

Lemma 2. *If $\alpha < \lambda$, then, for all $i \notin \mathcal{A}$, $\rho^{R_i} \geqslant \gamma'\beta^{R_i}/2$.*

Proof. See Section 5.10. □

These lemmas combine to prove Theorem 2, which states that either the malicious relay sends less than expected, putting it into probation, or it obtains a relative

consensus weight at most a constant factor above the relative amount of traffic it transfers. With $\alpha = 0$, $\gamma' = 1$, $\mu = 0.75$, $\epsilon_{\text{dec}} = 0.25$, $\epsilon_{\text{inc}} = 0.25$, and $\lambda = 0.256$, the advantage factor is $\gamma = 5.2$.

Theorem 1. *If* $\beta^A < (1-\lambda-\alpha)(1-1.4\epsilon_{dec})\mu\rho_0^A t^A/t_0^A$*, then the expected value of* ρ^A *puts* A *into probation. Otherwise,*

$$\frac{\mathbb{E}[\omega^A]}{\sum_R \omega^R} \leqslant \left[\frac{2\gamma'(1+\epsilon_{inc})(1-\epsilon_{dec})}{(1-\lambda-\alpha)(1-1.4\epsilon_{dec})\mu \sum_R}\right] \frac{(\beta^A/t^A)}{(\beta^R/t^R)}.$$

Proof. See Section 5.10. □

While in theory the adversary can obtain the advantage factor γ in Theorem 2 (the value in brackets), doing so requires performing certain attacks that have their own costs and limitations. To obtain the factor of 2 in γ, the adversary must only send or receive "one-sided" traffic to measuring relays, *i.e.* only from the client side or destination side. In particular, he must not actually relay any Tor traffic. Acting this way is highly observable and is likely to be noticed by clients. To obtain the $1/((1-\lambda-\alpha)\mu)$ factor in γ, the adversary must only exchange traffic with a set of measuring relays with voting weight $1-\lambda-\alpha$. This is observable by Directory Authorities (although hard to distinguish from a possible framing of an honest relay) and also noticeable by clients.

5.6.2 Weights across voting periods

We now consider that the ability of an adversary to increase his inferred capacities κ^A over time, $A \in \mathcal{A}$. For this analysis, we ignore the possible inflation effect of the added noise, assume that honest relays send their share of the traffic volume to the measuring relays (*i.e.* that $\gamma' = 1$), and assume that voting weights and selection

probabilities are maintained to be proportional (*e.g.* as when there is no network churn).

The adversary can apply a *feedback attack* to the bounds of Theorem 2 by repeatedly exchanging all traffic with a target set of relays with voting weight $1 - \lambda - \alpha$, including the trusted relays, as α grows. He can accomplish this, for example, by severely rate limiting honest client connections and creating his own dummy connections through the targeted measuring relays. Initially, when $\alpha = 0$, this inflates his total capacity by a factor of at most $2/((1-\lambda)\mu)$. Then, once the adversary receives an inflated voting weight α, he can target $1-\lambda-\alpha$ of the measuring relays to exchange all traffic with, further increasing his voting weight by a factor of $(1-\lambda)/(1-\lambda-\alpha)$. The adversary can repeat this process until either he reaches a fixpoint or he receives voting weight $\alpha > \lambda$, at which point the capacities of his relays are limited by the ceiling set by the trusted-relay measurements (and are unbounded without trusted relays), and the adversary can start to deflate the weights of the honest relays, which are ultimately limited by the floor set by the trusted-relay measurements.

We simulate this process across voting periods until the adversary's weight no longer increases. We also consider a simpler variant of PeerFlow in which *only* trusted-relay measurements are used. In this variant, the trusted-relays measurements are used directly (*i.e.* $\rho^R = \hat{\rho}_{\max}$), there are no voting weights, and the other PeerFlow components operate in the same way.

The results are shown in Figure 5.5. It shows that PeerFlow can provide bounded weight inflation to small adversaries even when there are *no* trusted relays. Specifically, the inflation factor is $\gamma < 4.6$ when $\tau = 0$ and the adversary is at most 4% of the network. It also shows that using measurements from all relays protects against small adversaries at the expense of worse security against larger adversaries compared to using measurements from only trusted relays. Making this tradeoff is

consistent with Tor's security in general, which can only protect against adversaries of small relative size [36]. We can also see that as the trusted fraction of the network grows, the security from using all relays and from using only trusted relays become the same. In both cases, as the trusted fraction grows, an adversary with ϕ of the network capacity has his inferred relative capacity limited to $(2\phi/\tau)/(1+2\phi/\tau)$, which implies a bounded inflation factor of $\gamma \leqslant 2/\tau$.

5.7 Load-Balancing Analysis

In this section, we experimentally demonstrate PeerFlow's ability to effectively distribute network traffic to relays in proportion to their bandwidth capacities.

5.7.1 Experimentation Setup

We evaluate PeerFlow and its effect on the consensus path selection weights, network goodput, and client performance using Shadow [33]. Shadow is an open-source parallel discrete-event network simulator/emulator hybrid that has been extensively validated and utilized for Tor network experimentation [31,32]. Shadow experiments are completely isolated from the network, so they are safe and contain no privacy risk to the public Tor network or its users. Because Shadow runs real applications as plug-ins, it is ideal for analyzing application-layer effects, *e.g.* those that result from modifying Tor's load-balancing protocols.

Our Private Tor Network We generated a new Tor network configuration for Shadow using the tools available in the Shadow distribution and Tor Metrics data [53] collected during 2014-09. The resulting Tor network contains 4 Tor directory authorities, 498 Tor relays, 7,500 Tor clients, and 1,000 servers. Our private network

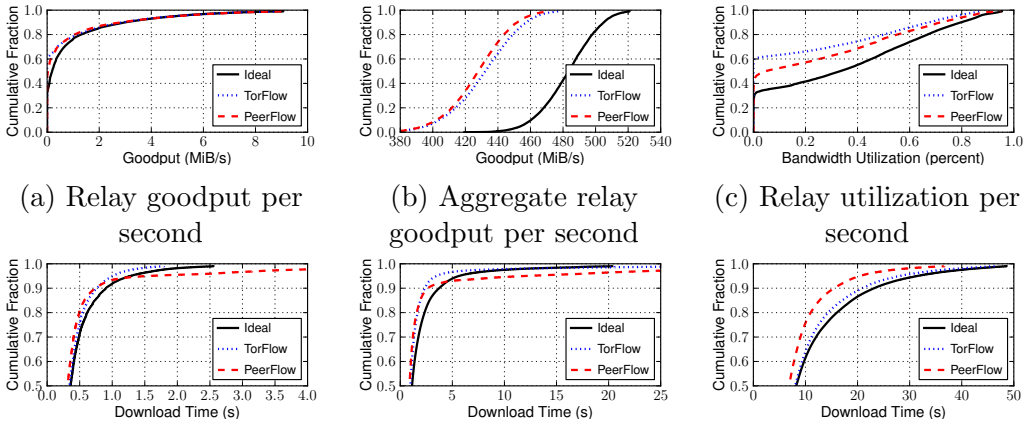
represents the public Tor network from 2014-09 at scale of approximately $\frac{1}{12}$, and we ran our private network for 5 virtual hours for each experiment (the first 2 of which were used for bootstrapping the network).

We experiment with and compare three models to determine the effect of load balancing weights on client performance and the distribution of network load. In the *Ideal* model, the relay consensus weights are initialized to their actual capacities as configured during the generation process, and the weights do not change throughout the experiment. In the *TorFlow* model, we run our new Shadow TorFlow plug-in that we developed[2] to mimic the behavior of TorFlow as it operates in the public Tor network. In the *PeerFlow* model, we run a PeerFlow prototype[3] that was implemented in the Tor code-base forked at version `0.2.5.10`. In both the *TorFlow* and *PeerFlow* models, the bandwidth information used to produce the consensus weights is dynamically updated throughout the experiment. Note that we do not compare EigenSpeed because it is not currently used in practice and because it is vulnerable to several attacks, as we have shown in Section 5.4, which we believe will prevent Tor from adopting it.

5.7.2 Network Performance

A primary goal of Tor's network load balancing algorithm is to distribute traffic to best utilize existing resources, and we use goodput as a metric for determining effective use of such resources. For our purposes, relay goodput, or application throughput, is the number of application bytes that a relay is able to forward over time. More formally, if a relay R_i receives $B_i^r(T)$ application bytes from the network and sends $B_i^s(T)$ application bytes to the network at time T over time interval t,

[2]Our Shadow TorFlow plug-in contains 2128 lines of code
[3]Our PeerFlow prototype contains 1222 lines of code



(a) Relay goodput per second

(b) Aggregate relay goodput per second

(c) Relay utilization per second

(d) Time to first byte of download

(e) Time to last byte of 320KiB download

(f) Time to last byte of 5MiB download

Figure 5.6: Network utilization and client performance for the *Ideal*, *TorFlow*, and *PeerFlow* load balancing models.

then we define its goodput as $G_i(T) = min(B_i^r(T), B_i^s(T))/t$. We normalize t to 1 second for ease of exposition.

To determine how load is distributed over the relays, we compute each relay's per-second goodput $G_i(T)$ for every second of the final 3 virtual hours of simulation. Figure 5.6a shows the cumulative distribution of the goodput for each relay-second, over all relays and seconds. The goodput achieved under all three models were 193.5 KiB/s for *Ideal*, 0.01 KiB/s for *TorFlow*, and 29.3 KiB/s for *PeerFlow* in the median, and 930.5 KiB/s for *Ideal*, 700.0 KiB/s for *TorFlow*, and 689.0 KiB/s for *PeerFlow* in the third quartile.

Given our definition of relay goodput, we define aggregate load L at time T as the sum of all relay goodputs, *i.e.*, $L(T) = \sum_i G_i(T)$. We compute L for all $t \in T$ (every second), and plot the cumulative distribution over all seconds in the experiment. The results are shown in Figure 5.6b. The median and standard deviation of aggregate goodput are 483.5 and 17.3 MiB/s for the *Ideal* model, 432.2 and 20.9 MiB/s for the *TorFlow* model, and 428.0 and 19.5 MiB/s for the *PeerFlow* model. Given these results, we conclude that PeerFlow does not dramatically reduce relay goodput

compared to the *TorFlow* model, and that an optimized implementation of PeerFlow may be able to increase aggregate relay goodput.

We also consider network utilization as a load balancing metric, which demonstrates how well the algorithms in our load balancing models use the available network resources. If relay R_i has goodput $G_i(T)$ and capacity C_i, we define its utilization $U_i(T)$ at time T as $U_i(T) = G_i(T)/C_i$. We compute utilization for each relay each second, and show the distribution of per-second utilization rates over all relays in Figure 5.6c. Our results show that 75% of relays had a utilization of 40.9% or less in *TorFlow*, 48.0% or less in *PeerFlow*, and 61.2% or less in *Ideal*. We thus conclude that while *PeerFlow* improves utilization over *TorFlow*, further improvements may still be possible.

5.7.3 Client Performance

Another goal of load balancing algorithms is to minimize client performance bottlenecks. We use file download times as a metric for determining how clients will perceive performance changes resulting from using PeerFlow. We consider the time to download the first byte of all files as a general measure of latency (Figure 5.6d), and the time to download the last byte of 320KiB files (Figure 5.6e) and 5MiB files (Figure 5.6f) as measures of performance for *web* or *bulk* type downloads. As the distribution of download times is nearly identical for all measurements below the median, we focus on the upper half of the distributions.

With regards to the 320KiB file downloads in Figure 5.6e, the 3rd quartile download time is highest for *Ideal* at 2.00 seconds, followed by *TorFlow* at 1.47 seconds, and lowest for *PeerFlow* at 1.36 seconds. The maximum download time is just over 60 seconds for all models due to the default 60 second timeout set by the traffic generation tool used in Shadow. These trends are similar for both 320KiB and 5MiB

	min	**q1**	**med**	**q3**	**max**	**stdev**
Ideal	21	176	617	2108	42280	4172
TorFlow	0	7	58	2801	737732	57957
PeerFlow	0	32	96	411	21289	2437

Table 5.4: Statistical summary of the raw relay weights from the ultimate Tor network consensus document.

downloads, as well as for the time to first byte metric.

Given our results, we conclude that the security benefits of PeerFlow come with a slight reduction in download times across all file sizes when compared to both *TorFlow* and *Ideal*, and a small increase in download time for the smaller first byte and 320KiB metrics for less than 10% of the downloads. In our incomplete prototype, because probation not implemented, it is possible that PeerFlow will under-estimate relays' weights. We now analyze the extent of such errors in weight calculations in our experiments.

5.7.4 Consensus Weight Errors

TorFlow and PeerFlow produce load balancing weights that get added to the Tor network consensus. These weights bias the relays chosen by clients when constructing circuits, thus affecting the distribution of client load. A statistical summary of raw weights from the ultimate consensus file produced in our experiments (*i.e.*, in convergence) is given in Table 5.4. But, Tor's path selection algorithm considers the ratio of a given relay weight to the sum of all of the relays' weights, not the raw weights directly. Ignoring additional position weights that are applied based on the ability to serve as a guard or exit relay, this fraction roughly approximates the percentage of total network load that will be directed to a relay.

A good load balancing scheme should distribute load proportional to the real capacities of the relays; in other words, a relay's fractional weight should match its

Error type	Formula	TorFlow	PeerFlow
Abs. Accuracy	$\sum_i \lvert \mathcal{W}_i - \mathcal{C}_i \rvert$	24.72	33.89
Rel. Accuracy	$\sum_i \lvert \frac{\mathcal{W}_i - \mathcal{C}_i}{\mathcal{C}_i} \rvert$	14872	10099
Abs. Precision	$\sum_i stdev(\mathcal{W}_i)$	0.21	0.05
Rel. Precision	$\sum_i \frac{stdev(\mathcal{W}_i)}{mean(\mathcal{W}_i)}$	178.96	55.15

Table 5.5: Total absolute and relative accuracy and precision errors over all relays R_i with true capacities $\mathcal{C}_i$ and weights $\mathcal{W}_i$ as estimated by TorFlow and PeerFlow.

fractional capacity. Any deviation of its fractional weight from its fractional capacity represents an error. More formally, let C_i and W_i be the bandwidth capacity and consensus weight of relay R_i, respectively. Then, a relay's fractional bandwidth capacity is $\mathcal{C}_i = C_i / \sum_j C_j$ and its fractional consensus weight is $\mathcal{W}_i = W_i / \sum_j W_j$.

We consider *accuracy* of the weights produced by the load balancing algorithms to help understand the efficacy of each approach. We define absolute accuracy error for relay R_i as the difference between the weight $\mathcal{W}_i$ as estimated by the load balancing algorithm and the true capacity $\mathcal{C}_i$, and relative accuracy error as the absolute accuracy error relative to the true capacity. Because accuracy does not capture a consistent under- or over-estimation of the weights, we additionally consider the *precision* of the weights. We define absolute precision error for a given relay as the standard deviation of that relay's weights over all consensuses produced in the experiment, and relative precision error as the absolute precision error relative to the mean of those weights. Multiple weight estimates are produced during the experiment, and we then sum the absolute value of these errors over all relays R to get the total error for each experiment. The results of this analysis are given in Table 5.5. This analysis shows that our PeerFlow prototype produced less error than TorFlow in its estimated weights for all but the absolute accuracy metric, and that our prototype tended to consistently underestimate the available capacity of relays. We suspect that PeerFlow's weight estimates could be improved through the

use of dummy traffic as specified in Section 5.5.5, which was not implemented in our prototype. Based on this weight error analysis, we conclude that PeerFlow would be a suitable alternative to TorFlow for relay weight estimation.

5.8 Speed and Efficiency Analysis

PeerFlow schedules measurement rounds individually for each relay depending on the relay's expected client traffic. With more client traffic, a relay's measurements will more quickly be large relative to the added noise, and so the measurement time can be smaller. These measurement times also affect the bandwidth efficiency of the protocol because they determine how often measurements are sent to the Directory Authorities.

5.8.1 Speed

In order to estimate relay measurement times when running PeerFlow on the Tor network, we use historical network data from CollecTor [11]. These data include past network consensuses, from which we can determine the types of relays that existed, and extra-info descriptors, from which we can determine the amount of traffic each relay transferred.

We use the consensus from January 31, 2015, at 00:00 and the extra-info descriptors from January 2015 as the basis for our analysis. We estimate the average rate of client traffic transferred by each relay using the byte histories over the hour before the consensus. After excluding relays with empty or incomplete byte histories or with selection probabilities of zero, there are 5917 relays in the network.

We estimate PeerFlow's determination $\bar{\rho}_p^R$ of the bytes relayed by R and observed in position p by taking the minimum of read and written bytes in the byte history

and multiplying the result by R's relative probability of being selected in each circuit position. We take the measuring guards, measuring middles, and measuring exits to be the largest $\mu = 0.75$ fraction of relays by position-weighted bytes relayed (*i.e.*, for relay R, ρ^R is multiplied by the position weight for selecting R in the given measuring position). The result is 400 measuring guards, 749 measuring middles, and 145 measuring exits. We classify as guard-type those relays that have a larger probability of selection as a guard. We classify the rest as exit-type. In Tor, relays can potentially serve as both guards and exits, and the bandwidth weights are set to balance the total capacity among guards, middles, and exits. However, in the consensus of January 31, 2015, at 00:00, the bandwidth weights are such that relays that have both the Guard and Exit flags are chosen as exits with probability 1, relays with only the Exit flag are chosen as exits with probability 1, and relays without the Exit flag have exit policies that are unlikely to allow them to serve much as an exit. Thus our analysis should not change under our division of relays into guard-type and exit-type.

We apply the measurement-time procedure given in Section 5.5.4 to estimate what the measurement times of existing Tor relays would be under PeerFlow. Figure 5.7 shows the resulting distribution of measurement times over the observed bytes transferred. Measurement times below 14 days are shown, which constitute 96.8% of the times by relay capacity. The 25th-percentile measurement time is 11.5 hours, the 50th is 27.7 hours, and the 75th is 70.7 hours. The relays with measurement times above 14 days have low observed capacity, many below Tor's stated requirements of 100 KB/s for Fast relays and 250 KB/s for guards. By raising these requirements by 150%, to 250 KB/s for Fast relays and 625 KB/s for guards, and excluding relays with observed capacities below these requirements, the maximum measurement time can be lowered to 316.4 hours (*i.e.* 13.2 days).

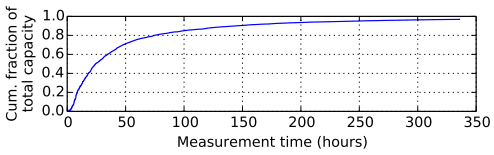


Figure 5.7: Measurement times weighted by observed relay capacity

5.8.2 Efficiency

PeerFlow has bandwidth overhead from sending statistics to the Directory Authorities and from any dummy traffic sent by relays in the PROBATION state. Relays should rarely enter PROBATION, which occurs if client traffic is unusually skewed away from the consensus weights, and so its cost should be small under normal operation. Each measuring relay sends data to the Directory Authorities at the end of a measured relay's measurement period. Thus the total amount of measurement traffic to them is $O(mn)$, where m is the number of measuring relays and n is the number of relays. However, each measuring relays sends only a few numbers for each measured relay, and our evaluation shows that measurement periods are generally at least a few hours long. Again using the consensus from January 31, 2015, at 00:00, and assuming that each statistic is 4 bytes, we find that the average rate of upload to each Bandwidth Authority is only 119.6 B/s.

5.9 Enhanced PeerFlow

While our simulations and analysis show that PeerFlow would improve the security and performance of Tor's load balancing at relatively low cost, we note that some future concerns are not addressed by our basic design. First, revealing the approximate amount of traffic sent between specific pairs of relays increases the information available to an adversary for traffic analysis. In addition, adding noise to traffic

statistics makes them less accurate and forces PeerFlow to wait to report on a given relay until enough client traffic is likely to have been relayed. This delays measurement of low-bandwidth relays, and as the network grows the relative weight of each measuring relay will decrease, thus increasing the length of reporting intervals.

Here we describe a modifications to PeerFlow that address these concerns, by cryptographically protecting the privacy of individual measurements. We present these modifications as an enhancement to PeerFlow to leave a core design that is more straightforward for the Tor Project to implement and adopt.

5.9.1 Encrypted Measurement Aggregation

In order to protect the privacy of individual measuring relay observations, we introduce a new role for the Bandwidth Authorities (Section 5.5.7). These authorities will have the role of jointly decrypting bandwidth reports after aggregation; if a majority of the Bandwidth Authorities collude then individual observations can be exposed, but if a majority are honest, then individual observation reports will retain their privacy.

The Bandwidth Authorities cooperate to publish a public key for a *threshold homomorphic tally system*, while separately maintaining shares of the decryption key. In addition to key generation, a threshold homomorphic tally system supports the following operations:

- *Encryption*: Given a public key and t tallies $b_1, b_2, \ldots, b_t$, produce a ciphertext that encodes these counts, providing semantic security.
- *Proving*: Given a public key PK, randomness r, and a ciphertext that encrypts a tally, produce a publicly-verifiable proof that the ciphertext is well-formed, *i.e.* was produced by computing $E_{PK}(0, \ldots, 1, \ldots, 0; r)$.

- *Aggregation*: Given ciphertexts c and c' encoding tallies $b_1, \ldots, b_t$ and $b'_1, \ldots, b'_t$ and weights w and w', produce a ciphertext c that encodes $wc_1 + w'c'_1, \ldots, wc_t + w'c'_t$.
- *Decryption*: Given ciphertext c and decryption key share s, produce a decryption share d along with a publicly-verifiable proof of correctness. The shares should be publicly combinable to produce the joint decryption. This verifiability allows PeerFlow to maintain auditability while increasing privacy.

We describe two possible implementations of these operations in Section 5.9.2 and 5.9.2.

Given the encrypted tally system, we modify how measuring relays report their observations. For each relay R, we partition the range $((1-\delta)\rho_p^R, (1+\delta)\rho_p^R)$ into t equally-sized buckets $[b_1, b_2), [b_2, b_3), \ldots, [b_{t-1}, b_t)$, where δ is a maximum allowed relative capacity change. Each measuring relay in position p then computes the observed value ρ_p^R (without noise) as in Section 5.5.1, and chooses the bucket $b \in \{1, \ldots, t\}$ containing ρ_p^R. The measuring relay R' then encrypts a "vote" for this bucket, and submits this encrypted tally $c^{R'}$ to the Bandwidth Authorities along with a proof of well-formedness.

Finally, the Bandwidth Authorities use the homomorphic properties of submitted ballots to combine the tallies, weighted by measuring relays' voting weights, obtaining a final tally for each relay observation. The Bandwidth Authorities decrypt this aggregate, providing proofs of correct decryption. The Bandwidth Authorities then compute the observation for relay R, $\bar{\rho}_p^R$, as the mean value of the votes after trimming λ voting weight from the smallest and largest buckets. They send this value to the Directory Authorities.

5.9.2 Example Threshold Homomorphic Tally Schemes

Paillier-based scheme

Baudron et al. [3] describe a homomorphic tally scheme based on the Paillier cryptosystem; the scheme can be modified for PeerFlow in a straightforward way. The Bandwidth Authorities engage in a distributed threshold Paillier key generation algorithm, as in [16, 17, 23], resulting in a Paillier public key N with generator g. Using the network consensus, all measuring relays can agree on a value M which is greater than the sum S of all voting weights, for example $M = 2^{\lceil \log_2 S \rceil}$. Then to encrypt a vote for bucket i, a measuring relay chooses $r \in_R \mathbb{Z}_N^*$ and computes $E_N(b_i) = g^{M^i} r^N \bmod N^2$.[4]

Note that in this scheme, if the sum of voting weight given to bucket i is v_i, we have $E_N(b_i)^{v_i} = E_N(v_i M^i)$ and $\prod_i E_N(v_i M^i) = E_N(\sum_i v_i M^i)$, so as long as we have all $v_i < M$, we can recover weights $v_1, ...v_t$ by representing the result of decryption in base M (and this will be particularly efficient if $M = 2^k$ for some k).

Proving in zero-knowledge that a ciphertext $c = E_N(M^i)$ correctly encrypts a particular bucket value M^i can be accomplished by proving knowledge of an N-th root of c/g^{M^i} using the Guillou-Quisquater scheme, which requires the prover to send two elements of $\mathbb{Z}_{N^2}$ and a 2ℓ-bit challenge for soundness level $2^{-\ell}$; this can be extended using the standard techniques of Cramer *et al.* [14] to prove that $\bigvee_{i=1}^{t} c = E_N(M^i)$ and converted into a noninteractive proof using the Fiat-Shamir heuristic. All together, this method of proving that a ciphertext is correct produces a proof of length $2t(2\log_2 N + \ell)$ bits, and can be verified with t modular exponentiations in $\mathbb{Z}_{N^2}$.

Proofs of correct decryption are provided by each Bandwidth Authority individ-

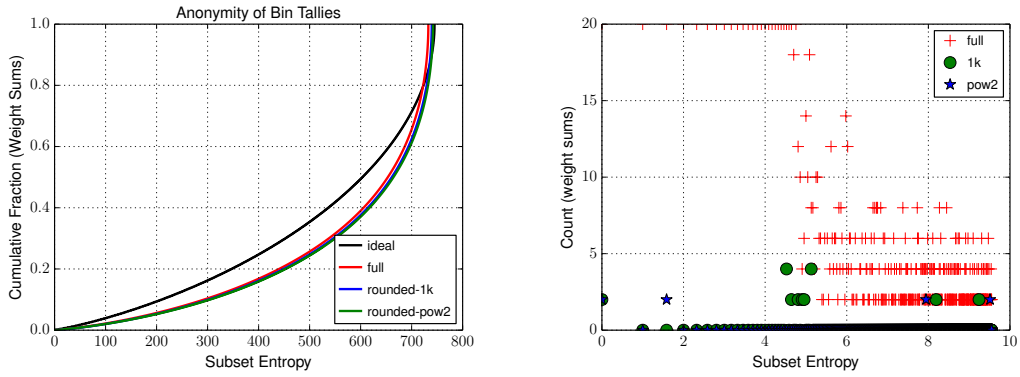
[4] If $M^t > N$, the scheme can use the Damgård-Jurik extension to work modulo N^{d+1}, where d is the minimal integer such that $M^t < N^d$.

ually, following the protocol described in [16, 17, 23].

Additive Elgamal-based scheme

A tally scheme can also be implemented using additive Elgamal encryption over any group where the Decisional Diffie-Hellman assumption is hard. In this case, the bandwidth authorities engage in a distributed key generation protocol to produce t public keys $\{h_i = g_i^x\}_{i=1}^t$ with generators $g_0, g_1, \ldots, g_t$ of prime order p whose respective discrete logarithms are unknown. To encrypt a vote for bucket b, a measuring relay chooses $r \in_R \mathbb{Z}_p$, and computes $c_0 = g_0^r$, $c_b = h_b^r g_0$, and $c_j = h_j^r$ for $j \neq b$; the ciphertext becomes the list $c_0, c_1, \ldots, c_t$. Note that in this scheme, raising all elements of a ciphertext to a scalar power v scales the tally for bucket b by v, and elementwise multiplication adds the tallies for all buckets. After decrypting the elements of a ciphertext using standard threshold Elgamal, the individual tallies are recovered in $O(\sqrt{\sum_i v_i})$ time using a standard short discrete logarithm algorithm.

Proving that a ciphertext c encrypts a vote for a single bucket b in honest-verifier zero knowledge could be accomplished as follows: the verifier chooses a random vector $\langle x_1, \ldots, x_t \rangle \in_R \mathbb{Z}_p^t$; then both sides compute $X = \sum_i x_i$, $H = \prod_i h_i^{x_i}$, and $C_b = g_0^{-x_b} \prod_i c_i^{x_i}$. Finally, the verifier and prover engage in a standard proof of knowledge of equality of discrete logarithms, $\log_{g_0} c_0 = \log_H C_b$, which requires the prover to send a group element and an element of $\mathbb{Z}_p$ and the verifier to send an ℓ-bit challenge. Applying the standard disjunction technique to the t possible buckets allows constructing a proof that c encrypts a single vote for one bucket $b \in \{1, \ldots, t\}$, and applying the Fiat-Shamir heuristic results in a noninteractive proof of length t group elements plus t elements of $\mathbb{Z}_p$ and t short exponents.



(a) Cumulative frequencies of subset anonymity

(b) Counts of low-anonymity subsets (capped at 20 to show detail)

Figure 5.8: The subset entropy of four voting-weight rounding schemes applied to the top 750 bandwidth weights from a recent Tor consensus: *full* is no rounding, *ideal* assigns all weights to 1, *rounded-1k* rounds to the nearest 1000, *rounded-pow2* rounds up to the nearest power of 2.

Privacy Analysis Under the assumption that the majority of Bandwidth Authorities are honest, this scheme ensures that no information is leaked about the individual measurements reported by each measuring relay beyond what is leaked by the voting-weighted sum released for a given measurement period; this follows from the semantic security of the tally encryption schemes. Here we analyze the amount of information that could be leaked by this aggregate result.

The residual leakage stems from the fact that few relays have identical weights in the current Tor network, so many small subsets of relays could have distinctive weight sums. As an example, we consider the bandwidth weights of the top 750 relays in the Tor network as of February 9, 2015 (750 is just over the number of measuring relays we used in our analysis Section 5.8.1). Among these relays, there are 138 possible sums that can only arise from a single subset of the weights, and 2456 possible sums that can arise from fewer than 256 different relay subsets.[5]

However, this leakage can be mitigated to a large extent by one of two schemes for

[5]To count possible subset sums, we modified the standard pseudopolynomial-time dynamic programming algorithm for subset sum, which runs in time $O(mS)$ for m items with total weight S

rounding of voting weights. In the *Rounded-1K* scheme, each measuring relay's weight is divided by 1000 and rounded. Continuing with the example above, we find that the 750th highest-weighted relay on Febuary 9, 2015 had a weight of 8160, which under this scheme would be assigned voting weight 8, while the highest-weighted relay had a weight of 334000, and would be assigned voting weight 334. In the *Rounded-Pow2* scheme, each measuring relay's weight is rounded to the next power of 2 and scaled by the voting weight of the lowest-weighted measuring relay. Thus the lowest-weighted measuring relay in the previous example would have its weight rounded to 8192 and then scaled to 1, while the highest-weighted relay's voting weight would be rounded to $2^{19} = 524288$ and scaled to 64. Figure 5.8 summarizes the impact of these rounding schemes on the anonymity of weight sums: overall, most relay subsets have high subset entropy (defined as $SE(R) = \log_2 |\{S \subseteq [n] : \sum_{i \in S} w_i = \sum_{i \in R} w_i\}|$), as seen in Figure 5.8a. As Figure 5.8b shows, however, both rounding schemes significantly reduce the number of subsets with low subset entropy, with *Rounded-Pow2* producing the fewest low-entropy subsets.

We note that in the worst case, the *Rounded-Pow2* scheme may inflate the fraction of voting weight controlled by an adversary by nearly a factor of 2, whereas the worst case for the *Rounded-1K* scheme is $1 + \frac{1}{2v_{\min}+1}$, where $v_{\min} \geqslant 1$ is the lowest post-rounding voting weight among measuring relays; in our example $v_{\min} = 8$, so the worst-case inflation factor as a result of rounding was 1.059. If we instead consider an adversary that compromises existing relays in our example, the highest relative inflation encountered under the *Rounded-Pow2* scheme was 1.366, while the highest relative inflation under the *Rounded-1K* scheme was 1.014; the respective 90th percentiles were 1.287 and 1.014.

5.10 Proofs of Theorems

Lemma 3. *If $\alpha < \lambda$, then*

$$\mathbb{E}[\rho^A] \leqslant \frac{\beta^A}{(1-\lambda-\alpha)\mu} + \frac{0.4\epsilon_{dec}\rho_0^A t^A}{t_0^A}.$$

Proof. The Directory Authorities infer the number of bytes transferred by A as the maximum of inferred bytes observed on the client and destination sides: $\rho^A = \max(\rho_c^A, \rho_d^A)$. The inferred number of client-side bytes is the sum of the inferred number of guard-observed and middle-client-side bytes: $\rho_c^A = \bar{\rho}_g^A + \bar{\rho}_{mc}^A$. The inferred number of destination-side bytes is defined similarly: $\rho_d^A = \bar{\rho}_e^A + \bar{\rho}_{md}^A$. Let β_p^A be the number of bytes sent or received by A with a measuring relay in position $p \in \{g, mc, md, e\}$. We will show show a bound on each $\bar{\rho}_p^A$ in terms of β_p^A that will combine to bound both ρ_c^A and ρ_d^A and then ρ^A as well.

$\bar{\rho}_p^A$ is determined from the values $\beta_p^{R_i A}$, which are the number of bytes A sent or received from each measuring relay $R_i \in \mathcal{M}_p$ while R_i was in position p in the circuit. Each R_i uses $\beta^{R_i A}$ to produce an independent estimate of the total bytes $\rho_p^{R_i A}$ by first adding random noise $N_p^{R_i A} \sim \mathsf{Lap}(\tau_n/\epsilon_n)$ and then dividing by its selection probability $q_p^{R_i}$, that is, $\rho_p^{R_i A} = (\beta_p^{R_i A} + N_p^{R_i A})/q_p^{R_i}$. $\bar{\rho}_p^A$ is the aggregate of these estimates after trimming the largest and smallest fraction λ by voting weight. Let i_j be the index of the jth largest ρ statistic. Let i_{j_1} and i_{j_2} be the indices of the first and last untrimmed statistics, respectively. Let $\mathcal{A}$ be the set of indices of the adversary's relays. Let N'^i be independent random variables from an exponential distribution $\mathsf{Exp}(\epsilon_n/\tau_n)$ ($\mathsf{Exp}(\lambda)$ has density function $f(x) = \lambda e^{-\lambda x}$). To extend the measuring relay notation to include directional positions, let $\mathcal{M}_{mc} = \mathcal{M}_{md} = \mathcal{M}_m$.

We can bound the value of $\bar{\rho}_p^A$ as follows:

$$\bar{\rho}_p^A = \frac{\sum_{j_1 \leqslant j \leqslant j_2} {\rho_p^{R_{i_j} A}}\, {q_p^{R_{i_j}}}}{\sum_{j_1 \leqslant j \leqslant j_2} {q_p^{R_{i_j}}}} \tag{5.1}$$

$$= \frac{\sum_{j_1 \leqslant j \leqslant j_2} {\beta_p^{R_{i_j} A}} + {N_p^{R_{i_j} A}}}{\sum_{j_1 \leqslant j \leqslant j_2} {q_p^{R_{i_j}}}} \tag{5.2}$$

$$\leqslant \frac{\sum_{j_1 \leqslant j \wedge j \notin \mathcal{A}} {\beta_p^{R_{i_j} A}} + {N_p^{R_{i_j} A}}}{\sum_{j_1 \leqslant j \wedge j \notin \mathcal{A}} {q_p^{R_{i_j}}}} \tag{5.3}$$

$$\leqslant \frac{\beta_p^A}{(1-\lambda-\alpha)\mu} + \frac{\sum_{j_1 \leqslant j \wedge j \notin \mathcal{A}} {N_p^{R_{i_j} A}}}{(1-\lambda-\alpha)\mu} \tag{5.4}$$

$$\leqslant \frac{\beta_p^A}{(1-\lambda-\alpha)\mu} + \frac{\sum_{1 \leqslant i \leqslant |\mathcal{M}_p|} N^{\prime i}}{(1-\lambda-\alpha)\mu} \tag{5.5}$$

The inequality in line 5.3 holds because $j > j'$ implies that ${\rho_p^{R_{i_j} A}} \geqslant {\rho_p^{R_{i_{j'}} A}}$ and also any untrimmed voting weight from adversarial relays is replaced by the same amount of voting weight from honest relays with larger ρ statistics. The inequality in line 5.4 holds because the voting weight of honest relays with ρ statistics above the lower trimmed fraction have weight $1-\lambda-\alpha$, with $\alpha < \lambda$, and voting weights are updated to maintain that the selection probability of any subset with weight at least $1-2\lambda$ has total selection probability of at least μ of its voting weight. The inequality in line 5.5 holds because "remapping" a negative noise value to its absolute value only increases its value and results in a random variable with distribution $\mathsf{Exp}(\epsilon_n/\tau_n)$, and then including additional such remapped random variables only increases the sum as they are all nonnegative.

Using this bound on $\bar{\rho}_p^A$, we can see that

$$\rho_c^A = \bar{\rho}_g^A + \bar{\rho}_{mc}^A \tag{5.6}$$

$$\leqslant \frac{\beta^A}{(1-\lambda-\alpha)\mu} + \frac{\sum_{1 \leqslant i \leqslant |\mathcal{M}_g|+|\mathcal{M}_m|} N'^i}{(1-\lambda-\alpha)\mu}. \tag{5.7}$$

By similar arguments, an analogous bound holds for ρ_d^A.

ρ^A is the maximum of ρ_c^A and ρ_d^A. Let $n_m = \max(|\mathcal{M}_g| + |\mathcal{M}_{mc}|, |\mathcal{M}_e| + |\mathcal{M}_{md}|)$, let $N_c'^i$ and $N_d'^i$ be independent random variables with distribution $\mathsf{Exp}(\epsilon_n/\tau_n)$, and let $S_p = \sum_{1 \leqslant i \leqslant n'} N_p'^i$, $p \in \{c, d\}$. Then

$$\begin{aligned}\mathbb{E}[\rho^A] = \mathbb{E}[\max(\rho_c^A, \rho_d^A)] \\ \leqslant \mathbb{E}\Bigg[\frac{1}{(1-\lambda-\alpha)\mu} \max\big(\beta_g^A + \beta_{mc}^A + S_c, \\ \beta_e^A + \beta_{md}^A + S_d\big)\Bigg]. \end{aligned} \tag{5.8}$$

This inequality follows using the same reasoning that established the inequality in line 5.5.

For a fixed bandwidth budget β^A, the right-hand side of inequality 5.8 is maximized by setting one of $\beta_g^A + \beta_{mc}^A$ and $\beta_e^A + \beta_{md}^A$ to β^A and the other to 0. The follows from the fact the sums of the noise variables S_p are identically distributed, and so every case in which the maximum is reduced corresponds to a unique and equally-probably case in which the maximum in increased by an equal or larger amount.

Therefore,

$$\mathbb{E}[\rho^A] \leqslant \frac{1}{(1-\lambda-\alpha)\mu}\mathbb{E}\left[\max(0+S_c, \beta^A + S_d)\right] \tag{5.9}$$

$$\leqslant \frac{\beta^A + 2\mathbb{E}[S_d]}{(1-\lambda-\alpha)\mu} = \frac{\beta^A + 2n'\tau_n/\epsilon_n}{(1-\lambda-\alpha)\mu} \tag{5.10}$$

$$\leqslant \frac{\beta^A}{(1-\lambda-\alpha)\mu} + \frac{0.4\epsilon_{\text{dec}}\rho_0^A t^A}{t_0^A}, \tag{5.11}$$

where the inequality in line 5.11 holds because t^A was set in order to make it true. □

Lemma 4. *If $\alpha < \lambda$, then, for all $i \notin \mathcal{A}$, $\rho^{R_i} \geqslant \gamma'\beta^{R_i}/2$.*

Proof. The adversary can only lower ρ^{R_i} if he reduces the ρ statistics from his relays, and $\alpha < \lambda$ so we can assume the set of measurements in the trimmed sum are from some set of honest relays of voting weight $1-2\lambda$. The aggregate of these measurements in a given position is at least $\gamma'\beta_p^{R_i}$. Because honest relays send each byte that they receive and vice versa, $\beta_g^{R_i} + \beta_{mc}^{R_i}$ or $\beta_e^{R_i} + \beta_{md}^{R_i}$ is at least $\beta^{R_i}/2$. Thus

$$\begin{aligned}
\rho^{R_i} &= \max(\bar{\rho}_g^{R_i} + \bar{\rho}_{mc}^{R_i}, \bar{\rho}_e^{R_i} + \bar{\rho}_{md}^{R_i}) \\
&\geqslant \gamma' \max(\beta_g^{R_i} + \beta_{mc}^{R_i}, \beta_e^{R_i} + \beta_{md}^{R_i}) \\
&= \gamma'\beta^{R_i}/2.
\end{aligned}$$

□

Theorem 2. *If $\beta^A < (1-\lambda-\alpha)(1-1.4\epsilon_{dec})\mu\rho_0^A t^A/t_0^A$, then the expected value of ρ^A puts A into probation. Otherwise,*

$$\frac{\mathbb{E}[\omega^A]}{\sum_R \omega^R} \leqslant \left[\frac{2\gamma'(1+\epsilon_{inc})(1-\epsilon_{dec})}{(1-\lambda-\alpha)(1-1.4\epsilon_{dec})\mu\sum_R}\right]\frac{(\beta^A/t^A)}{(\beta^R/t^R)}.$$

Proof. If $\beta^A < (1-\lambda-\alpha)(1-1.4\epsilon_{\text{dec}})\mu\rho_0^A t^A/t_0^A$, then, by Lemma 1, $\mathbb{E}[\rho^A] < (1-$

$\epsilon_{\text{dec}})\rho_0^A t^A / t_0^A$. ω^A is adjusted to keep ρ^A no less than $(1 - \epsilon_{\text{dec}})\eta^R$, and thus the expected value of ρ^A would put A into probation.

Otherwise, we can simply substitute the right-hand side of this inequality for β^A into the inequality of Lemma 1 to yield

$$\mathbb{E}[\rho^A] \leqslant \frac{(1 - \epsilon_{\text{dec}})\beta^A}{(1 - 1.4\epsilon_{\text{dec}})(1 - \lambda - \alpha)\mu}.$$

Then we can use Lemma 4 and the fact that ω^A is set to at most $(1 + \epsilon_{\text{inc}})\rho^A / t^A$ to obtain the theorem. □

Chapter 6

Proof of Security of Verdict

6.1 Introduction to Verdict

The previous chapter concerned improvements to the network behind Tor, an extremely popular anonymity protocol based on onion routing. Tor's greatest advantage is its speed, and ability to function even in an environment requiring low latency. However, Tor's onion routing approach to anonymity has some inherent weaknesses. An adversary capable of observing a large enough part of the network can associate users with their messages, as can an adversary controlling a substantial fraction of Tor relays [4, 36]. Malicious users and relays are also difficult to detect and identify [58] in Tor.

Some protocols address these issues by using core technologies other than onion routing. Dining-cryptographers networks protocols, or DC-nets, provide a basis for a type of anonymity protocol that *provably maintains anonymity* against adversaries with the ability to read all messages sent over the network, and against coalitions of all except for a few honest participants [10, 25]. Two such protocols include Dissent [12] and Verdict [13]. These protocols also offer *accountability*, in the sense that

honest parties can detect when another party deviates from the protocol, and *integrity*, meaning that honest parties will detect any tampering with the anonymous messages of other honest parties.

In this chapter, we examine the Verdict protocol, which was introduced and implemented but not given a formal specification in [13]. In Section 6.2, we provide a full and formal description of the Verdict protocol for the first time.

In Section 6.3, we define the properties of a DC-nets protocol and of a verifiable shuffle protocol. Our description follows [13] in treating these underlying protocols as black boxes, allowing for different choices of internal protocol to be used for different applications. Three examples of DC-nets protocols that satisfy our requirements are given in [13]. We also specify, in Section 6.3.3, formal definitions for the properties we want our anonymity protocol to satisfy: *anonymity*, *accountability*, and *integrity*. Finally, in Section 6.4, we provide a formal proof that, given a DC-nets protocol and a verifiable shuffle protocol, Verdict satisfies these three properties.

6.2 The Verdict Protocol

This section defines the core `Verdict` protocol, as well as several sub-protocols and methods that Verdict uses.

6.2.1 Assumptions and Architecture

All protocols are run between N clients and M servers. Each client directly communicates with at least one upstream server, and each server directly communicates with all other servers.

Every party i has a long-term private signing key v_i and a public verification key V_i for an agreed-upon cryptographic signature scheme. We assume that all parties

have access to the functions $\sigma \leftarrow \texttt{Sign}(v_i, m)$ and $VER \leftarrow \texttt{Verify}(V_i, m, \sigma)$. Every party also has access to a common oracle, $\mathcal{O}$. $\mathcal{O}$ can be a random oracle, or it can be an oracle which returns a common reference string, as specified by the non-interactive zero-knowledge proof-of-knowledge system used by the DC-nets construction.

For a party to participate in the protocol, every other party that will be participating must have access to its public verification key V_i. Before the protocol begins, all parties are aware of a list of participating servers, and a list of potentially participating clients. These groups are agreed upon before beginning the protocol, and we do not discuss the details of how the groups are chosen here. We note, however, that every server in the group is assumed to be online and participating in the protocol, but not every client in the group is assumed to be online. Messages from any party not in the groups are ignored during all of the following protocols. In addition, in all protocols except for `SetupA`, parties ignore messages from clients for whom they do not possess a verified and valid client public key (in other words, they ignore messages from clients which are in the group but did not participate in `SetupA`).

We assume the parties communicate using asynchronous message passing. A party *broadcasts* a message to a specified set by sending the same message individually to all parties in that set. We do not handle non-responsive parties; therefore, none of the following protocols are guaranteed to terminate.

When a party detects a failure, whether it causes the party to halt or not, it produces a proof $\pi = (V_i, f, P, mess_i)$ blaming the party i that caused failure f, with the data in P as the evidence for the proof. The proof also includes the set $mess_i$ of all protocol messages sent by i sent under the current nonce that the party constructing the proof has access to. This set of protocol messages is needed to verify that i actually transmitted the data in P which is being used to blame it. A separate method called `ProofVer` (described in section 6.2.6) allows a third party to verify a

blame proof.

We do not assume that the channels are private, authenticated, or tamper-proof. That is, the channel itself does not prevent other parties from viewing, altering, or dropping messages, nor does it prevent them from inserting messages between two parties. Our design instead uses signatures, nonces, and step numbers to provide some of this security, as described below.

With each message sent over the network, it is important for the sender to include some additional information along with it. In these protocols, when a party sends the message m, the data sent is actually the tuple (m, V_i, n, t, σ). m is the payload, or the intended message, and the rest of the information is referred to as the *metadata*. V_i is the sender's public verification key and also serves as the sender's "identity". n is the current nonce. Unless specified otherwise, the current nonce is either the session nonce n_S, the round nonce n_R, or the slot nonce n_S, whichever was most recently set by the protocol. t is a number identifying the protocol and step under which this message was sent. The values n and t together serve to prevent replay attacks. Finally, the metadata includes a signature $\sigma = \texttt{Sign}(v_i, (m, n, t))$.

Whenever a party's message contains any value generated by a third party, such as a ciphertext, the metadata must also include the verification tuples that accompanied each of those values. This will mean that to send or forward a message, the identity of the sender in the form of its public key, and the nonce and signature on that message, will always be attached to it. This ensures that messages can be verified by third parties at a later date. A protocol message which does not contain a verification tuple is dropped by the receiver.

Finally, we assume that all parties know two agreed-upon parameters: ρ, a security parameter; and R, the number of rounds for which the protocol will execute.

6.2.2 Setup Protocols

There are two protocols used for session setup: `SetupA` and `SetupB`. `SetupA` is for selection and of nonces and session keys and dissemination of public keys, while `SetupB` is for further setup dependent on the implementation of the underlying DC-nets ciphertext system.

`SetupA`

$(ERR, data_A, \pi_i) \leftarrow$ `SetupA` (ρ, v_i) takes the security parameter ρ and each party's private signing key v_i as input. For each party, it outputs error status ERR, which is either SUCCESS or FAIL, and some data, which may include a list of third-party-verifiable proofs blaming parties which have sent invalid keys. `SetupA` will only fail if a server sends or forwards an invalid key, but its list of blame proofs will include both servers and clients who have sent invalid keys.

For a client i, if $ERR =$ SUCCESS, $data_A = (n_S, A_i, a_i, Y_i, y_i, \mathbf{B})$. That is, the output consists of a session nonce n_S identifying a run of `Verdict`, a client public key A_i, a client private key a_i, a pseudonym public key Y_i, a pseudonym private key y_i, and the set of server public keys $\mathbf{B}$. If $ERR =$ FAIL, $data_A$ is blank. In either case, the output of `SetupA` also includes π_i, a possibly-empty set of blame proofs.

For a server j, if $ERR =$ SUCCESS, $data_A = (n_S, B_j, b_j, \mathbf{A}, \mathbf{B})$. This data includes the session nonce n_S, a server public key B_j, a server private key b_j, a set of client public keys $\mathbf{A}$, and the set of server public keys $\mathbf{B}$. If $ERR =$ FAIL, $data_A$ is blank. Again, π_i is an output in either case.

`SetupA` handles the setup steps which will be needed regardless of the choice of DC-nets ciphertext system used by the rest of the protocol, but some of these steps use methods which must be provided by the ciphertext system itself, specifically `KeyGen` and `VerifyKey`, both of which are described in section 6.2.4.

Given these methods, `SetupA` does the following:

1. The servers collaboratively select a session nonce n_S, then broadcast n_S to their downstream clients. As long as all servers agree on a reliable method to jointly select a nonce, it does not matter which method is used, but the following is presented as an example:

   (a) Each server j selects a unique random string x_j and broadcasts it to all other servers. For this message, the nonce part of the metadata is blank.

   (b) Upon receipt of string x_k from server k, server j sends x_j to k again. For this message, the nonce part of the metadata is x_k. (Note that the metadata also contains a signature on the message and the nonce.)

   (c) Once server j has received the same x_k from server k, with x_j used as the nonce, it concatenates all random strings received from other servers together in a pre-arranged way (e.g. sorted by public verification key) to get n_S, the nonce.

2. Each client i runs $(A_i, a_i, PoK_i) \leftarrow \texttt{KeyGen}(\rho)$ and $(Y_i, y_i) \leftarrow \texttt{KeyGen}(\rho)$ to get a client keypair (A_i, a_i) and pseudonym keypair (Y_i, y_i). Client i sends (A_i, PoK_i) to its upstream server(s).

3. Each server j runs $VER \leftarrow \texttt{VerifyKey}(A_i, PoK_i)$ upon receiving a message from client i. If VER is False for client i's message, the server adds $(V_i, \mathsf{f_1}, A_i, PoK_i, mess_i)$ to π_j.

4. Each server j collects the set of client public keys from its downstream clients for which VER was True into $\mathbf{A}_j$, and the set of associated proofs of knowledge into $\mathbf{PoK}_j$.

5. Each server j runs $(B_j, b_j, PoK_j) \leftarrow \texttt{KeyGen}(\rho)$ to get a server keypair (B_j, b_j) and proof of knowledge PoK_j. Server j then broadcasts $(B_j, PoK_j, \mathbf{A}_j, \mathbf{PoK}_j)$ to the other servers.

6. Upon receiving a message from server k, server j runs $VER \leftarrow \texttt{VerifyKey}(B_k, PoK_k)$. If $VER =$ False, it adds $(V_k, \mathsf{f_1}, B_k, PoK_k, mess_k)$ to π_j. j also re-runs $VER \leftarrow \texttt{VerifyKey}(A_i, PoK_i)$ for each public key and PoK pair in $(\mathbf{A}_k, \mathbf{PoK}_k)$. If $VER =$ False for any of these checks, j adds $(V_k, \mathsf{f_2}, \mathbf{A}_k, \mathbf{PoK}_k, mess_k)$ to π_j. If either of the verifications in this step failed, j outputs (FAIL, π_j) and halts. (Although the cause of failure was a faulty client key, j fails if server k were honest, it would not have forwarded that key.)

7. Each server j compiles all public keys received into $\mathbf{A} \leftarrow \cup_k \mathbf{A}_k$ and $\mathbf{B} \leftarrow \cup_k B_k$, and compiles all proofs of knowledge from the servers into $\mathbf{PoK}$. If $\mathbf{A}$ includes two different public keys A_i, A_i' associated with the same V_i (as determined by the metadata), j adds $(V_i, \mathsf{f_3}, A_i, A_i', mess_i)$ to π_j and removes both public keys from $\mathbf{A}$. It then sends $\mathbf{B}$ and $\mathbf{PoK}$ to its downstream clients, then halts and outputs $(\mathsf{SUCCESS}, (n_S, B_j, b_j, \mathbf{A}, \mathbf{B}), \pi_j)$.

8. Client i receives $\mathbf{B}$ and $\mathbf{PoK}$ from its upstream server(s) j. For each server k in the set of servers, i runs $VER \leftarrow \texttt{VerifyKey}(B_k, PoK_k)$. If $VER =$ False for any of these checks, i adds $(V_j, \mathsf{f_2}, B_k, PoK_k, mess_j)$ to π_i and outputs (FAIL, π_i).

9. If i has multiple upstream servers, and receives two different public keys B_j, B_j' associated with the same V_j, i adds $(V_j, \mathsf{f_3}, B_j, B_j', mess_j)$ to π_i and outputs (FAIL, π_i). Otherwise, it outputs $(\mathsf{SUCCESS}, (n_S, A_i, a_i, Y_i, y_i, \mathbf{B}), \pi_i)$.

`SetupB`

The second setup protocol, `SetupB`, takes the output of $data_A$ as input, and outputs $(ERR, data_B, \pi)$, where ERR is either $\mathsf{SUCCESS}$ or FAIL, and π is a possibly empty set of blame proofs. The contents of $data_B$ if $ERR = \mathsf{SUCCESS}$, and the functionality of the protocol otherwise, are entirely implementation-dependent. All proofs in π must have $f = \mathsf{f_4}$.

6.2.3 Verifiable Shuffles

A verifiable shuffle `VShuffle` is a protocol run between all clients and all servers. Each client i runs $(ERR, [\mathbf{m}], \pi_i) \leftarrow \texttt{ClientVShuffle}(v_i, \mathbf{B}, m_i)$, and each server j runs $(ERR, [\mathbf{m}], \pi_j) \leftarrow \texttt{ServerVShuffle}(v_j, b_j, \mathbf{B})$. We do not go into detail about verifiable shuffle protocols in this section, but we can describe the inputs and outputs of both the client-side and server-side protocols.

In `ClientVShuffle`, each client i encrypts its message m_i using the server public keys in $\mathbf{B}$, then sends this encrypted message to its upstream server(s). It then waits for the servers to communicate with each other, eventually receiving a response. Based on this response, the output of i is either $(\mathsf{SUCCESS}, [\mathbf{m}], \pi_i)$, where $[\mathbf{m}]$ is a permutation of the clients' messages and π_i is a possibly-empty set of blame proofs, or $(\mathsf{FAIL}, \bar{m}, \pi)$, where $\bar{m}$ is unreliable data, and π_i is a set of blame proofs which contains at least one proof. Proofs in π_i blaming party k must have the format $(V_k, \mathsf{f_5}, P, mess_k)$. The details of P, the evidence blaming k, are dependent upon the implementation of the verifiable shuffle.

In `ServerVShuffle`, each server j uses its own server private key b_j and the set of server public keys $\mathbf{B}$ to decrypt and shuffle the messages. When the messages have been shuffled fully, the server's output is either $(\mathsf{SUCCESS}, [\mathbf{m}], \pi_j)$ or $(\mathsf{FAIL}, \bar{m}, \pi_j)$.

Server j sends the second half of this message to its clients.

6.2.4 DC-nets Ciphertext Systems

A *DC-nets ciphertext system* is a set of eight methods. The methods are called `KeyGen`, `VerifyKey`, `CoverCreate`, `OwnerCreate`, `ClientVerify`, `ServerCreate`, `ServerVerify`, and `Reveal`. A DC-nets ciphertext system also specifies a special input *data*, an externally-generated value which is needed by some of the methods. Depending on the implementation, *data* might be a slot nonce n_T; the output of `SetupB` $data_B$, or blank. These methods also require access to the oracle $\mathcal{O}$; we omit $\mathcal{O}$ from the individual method descriptions for clarity of notation. Additionally, the `Verify` methods must be deterministic.

- $(K, k, PoK) \leftarrow$ `KeyGen`(ρ) takes as input a security parameter ρ. It returns a new public key K, associated private key k, and a proof of knowledge PoK certifying that K is a valid public key.

- $VER \leftarrow$ `VerifyKey`(K, PoK) takes as input a key public K annd a proof of knowledge PoK. The return value VER is True if public key K can be certified valid by PoK, and False if not.

- $(C_i, PoK_i) \leftarrow$ `CoverCreate`$(data, a_i, \mathbf{B})$ takes as input the implementation dependent *data*, i's client private key a_i, and the set of server public keys $\mathbf{B}$. It returns a client ciphertext C_i with no message, representing "cover traffic", and a proof of knowledge PoK_i certifying that C_i is valid.

- $(C_i, PoK_i) \leftarrow$ `OwnerCreate`$(data, a_i, y, \mathbf{B}, m)$ takes as input the implementation dependent *data*, a client private key a_i, the slot owner's pseudonym secret key y, the set of server public keys $\mathbf{B}$, and a plaintext message m. It returns

a client ciphertext C_i encoding message m, and a proof of knowledge PoK_i certifying that C_i is valid.

- $VER \leftarrow \texttt{ClientVerify}(data, A_i, \mathbf{B}, C_i, Y, PoK_i)$ takes as input the implementation dependent $data$, a client public key A_i, the set of server public keys $\mathbf{B}$, a client ciphertext C_i, the current pseudonym public key Y, and a proof of knowledge PoK_i. The return value VER is True if C_i and PoK_i were able to be verified, and False if not.

- $(S_j, PoK_j) \leftarrow \texttt{ServerCreate}(data, b_j, \mathbf{A}, \mathbf{C})$ takes as input the implementation dependent $data$, a server private key b_j, the set of client public keys $\mathbf{A}$, and a set of client ciphertexts $\mathbf{C}$. It outputs a server ciphertext S_j and a proof of knowledge PoK_j certifying that S_j is valid.

- $VER \leftarrow \texttt{ServerVerify}(data, \mathbf{A}, B_j, \mathbf{C}, S_j, PoK_j)$ takes as input the implementation dependent $data$, the set of client public keys $\mathbf{A}$, a server public key B_j, a set of client ciphertexts $\mathbf{C}$, a server ciphertext S_j, and a proof of knowledge PoK_j. The return value VER is True if S_j and PoK_j were able to be verified, and False if not.

- $m \leftarrow \texttt{Reveal}(\mathbf{C}, \mathbf{S})$ takes as input a set of client ciphertexts $\mathbf{C}$ and a set of server ciphertexts $\mathbf{S}$. It returns the plaintext message m.

6.2.5 Verdict

The `Verdict` protocol is run between N clients and M servers. Each session begins with the two setup protocols and a verifiable shuffle, then proceeds in *rounds*, each of which is broken up into N *slots*, one for each pseudonymous client. The protocol proceeds for the agreed-upon number of rounds R, but parties can halt earlier than

this if they detect a failure.

Failure Messages

When a party detects a failure during `Verdict`, whether it causes the party to halt or not, it produces a proof $\pi = (V_i, f, P, mess_i)$, as described in section 6.2.1. The party adds π to its output. If π blames a server, the party sends π to all of its neighbors in place of the next message it was send to them, and then halts. Otherwise, it sends π along with the next message it is to send.

Whenever a party receives a failure message π from another party, it runs $VER \leftarrow$`ProofVer`(π). If $VER =$`True`, the party adds π to its output. If the party receiving this message was a client, it then halts.

If a server receives and verifies a failure message π, it may need to pass along that message to other parties which may not have seen it. If the message comes from a client, it broadcasts π to the other servers in place of the next message it was supposed to send. If π comes from a server and blames a server, it sends π to its downstream clients in place of the next message it was to send to them. In either case, if π blamed another server, it then halts. If π blamed a client, the server removes that client from its list of active clients before continuing with the next step of the protocol as normal.

Detailed Verdict Protocol

A detailed description of `Verdict` is given below.

1. $(ERR, data_A, \pi_i) \leftarrow$ `SetupA` (ρ, v_i) is run by all servers and clients. Party i broadcasts π_i (even if π_i is empty), and adds π_i to its output. $ERR =$ `FAIL`, i then halts.

2. $(ERR, data_B, \pi_i) \leftarrow$ `SetupB` $(data_A)$ is run by all servers and clients. Party i broadcasts π_i (even if π_i is empty), and adds π_i to its output. $ERR = \mathsf{FAIL}$, i then halts.

3. $(ERR, [\mathbf{Y}], \pi_i) \leftarrow$ `ClientVShuffle`$(v_i, \mathbf{B}, Y_i)$ is run by each client i, and $(ERR, [\mathbf{Y}], \pi_j) \leftarrow$ `ServerVShuffle`$(v_j, b_j, \mathbf{B})$ is run by each server j. Party i broadcasts π_i (even if π_i is empty), and adds π_i to its output. $ERR = \mathsf{FAIL}$, i then halts. Otherwise, all parties treat $[\mathbf{Y}]$ as a shuffled list of pseudonym public keys. (It is possible that one or more clients submitted an invalid pseudonym public key, but as this will only have the effect of preventing that client from transmitting, there is no need to check for this.)

4. The initial round nonce, n_R, is set to n_S, the session nonce.

5. All parties perform the following outer loop R times.

   (a) All parties perform the following inner loop $|[\mathbf{Y}]|$ times, each time initializing the value Y to be the next pseudonym public key in the shuffled list $[\mathbf{Y}]$, starting with the first. In the following steps, the "slot owner" is the client who submitted Y as an input to `ClientVShuffle`.

      i. A slot nonce n_T is created by concatenating the round nonce n_R with the slot number.

      ii. $(C_i, PoK_i) \leftarrow$ `CoverCreate`$(data, a_i, \mathbf{B})$ is run by each client i other than the slot owner to generate a client ciphertext with a blank message and an associated proof of knowledge.

      iii. $(C_i, PoK_i) \leftarrow$ `OwnerCreate`$(data, a_i, y, \mathbf{B}, m)$ is run by the slot owner to generate a client ciphertext with message m and an associated proof of knowledge.

iv. Every client i sends (C_i, PoK_i) to its upstream server(s).

v. $VER \leftarrow \texttt{ClientVerify}(data, A_i, \mathbf{B}, C_i, PoK_i)$ is run by each server j on the message received from every client i. For each client whose message fails to verify (that is, $VER = \mathsf{False}$), the server j adds $(V_i, \mathsf{f_6}, data, A_i, \mathbf{B}, C_i, PoK_i, mess_i)$ to π_j.

vi. Each server broadcasts $\mathbf{C}_j$, the set of verified client ciphertexts received by j, along with all proofs added to π_j in the previous step, to all other servers.

vii. $VER \leftarrow \texttt{ClientVerify}(data, A_i, \mathbf{B}, C_i, PoK_i)$ is run by each server on all new client ciphertexts passed along by server j. If VER is False for any of these, the server broadcasts $(V_j, \mathsf{f_7}, data, A_i, \mathbf{B}, C_i, PoK_i, mess_i)$. It then halts.

viii. Each server j computes $\mathbf{C} = \cup_k \mathbf{C}_k$, combining the sets of client ciphertexts received by all servers into a single set. If $\mathbf{C}$ contains two different ciphertexts C_i and C_i' from the same client i, the server broadcasts adds $(V_i, \mathsf{f_8}, C_i, C_i', mess_i)$ to π_j. The server then broadcasts the set of all proofs created this way, or an empty set if no such proofs were created.

ix. Each server j removes from $\mathbf{C}$ all ciphertexts belonging to clients no longer on its list of active clients (because a proof blaming such a client was generated or validated by j).

x. $(S_j, PoK_j) \leftarrow \texttt{ServerCreate}(data, b_j, \mathbf{A}, \mathbf{C})$ is run by each server j to compute a server ciphertext and associated PoK from the set of client ciphertexts. Each server then broadcasts (S_j, PoK_j) to all other servers.

xi. $VER \leftarrow \texttt{ServerVerify}(data, \mathbf{A}, B_k, \mathbf{C}, S_k, PoK_k)$ is run by each server j to verify the ciphertext from server k. If VER is False for the ciphertext sent by k, j broadcasts $(V_k, \mathsf{f_9}, data, \mathbf{A}, B_k, \mathbf{C}, S_k, PoK_k, mess_k)$ and halts.

xii. Each server computes $\mathbf{S} = \cup_k S_k$, combining all server ciphertexts into a single set.

xiii. $m \leftarrow \texttt{Reveal}(\mathbf{C}, \mathbf{S})$ is run by each server to reveal the plaintext message m from the complete set of client and server ciphertexts.

xiv. $\sigma_j \leftarrow \texttt{Sign}(v_j, (m, N_T))$ is run by each server to create a signature on m along with the current slot nonce. Each server then broadcasts σ_j to all other servers.

xv. $VER \leftarrow \texttt{Verify}(V_k, (m, N_T), \sigma_k)$ is run by each server j to verify the signatures provided by every other server k. If VER is False for any signature, j broadcasts $(V_k, \mathsf{f_{10}}, m, N_T, \sigma_k, mess_k)$ and halts. Otherwise, j sends a blank message to the other servers to confirm that it is ready to distribute m to its clients.

xvi. Each server sends m and $\sigma \cup_k \sigma_k$ to each of its connected clients.

xvii. $VER \leftarrow \texttt{Verify}(V_k, m, \sigma_k)$ is run by each client to verify the signatures from every server k, as forwarded by each of its upstream servers j. If VER is False for any signature forwarded by server j, i broadcasts $(V_j, \mathsf{f_{11}}, V_k, m, \sigma_k, mess_j)$ and halts. Otherwise, the client sends a message to its upstream servers confirming successful receipt of m, and it adds m to its output.

xviii. Each server, after receiving confirmation from each client connected to it, every other server a message confirming that the slot has ended

successfully.

xix. Each server, after receiving confirmation from each server connected to it, sends its downstream clients a similar confirmation message and adds m to its output. (These confirmation messages simply allow the servers and clients an opportunity to forward any blame proofs generated in the final steps.)

(b) At the end of a round, the round nonce n_R is incremented by 1.

At the end of a session, a party's output has the format $(ERR, \mathbf{m}, \pi)$. $ERR =$ SUCCESS if the protocol ran for all R rounds without halting, and $ERR =$ FAIL if the party halted at any point before the end. The list $\mathbf{m}$ consists of all messages m recovered at the end of each completed slot, and π consists of all blame proofs that the party either constructed, or received and verified with `ProofVer`.

6.2.6 Blaming and Proof Verification

In Verdict and its associated protocols, a malicious client or server may cause a protocol to fail by sending invalid data. Whenever a party detects that the protocol has failed or determines that another party has sent invalid data, it produces a blame proof. Such proofs are tuples π with the format $(V_i, f, P, mess_i)$, where V_i is the public verification key of the party i being blamed, f is a code representing the type of failure for which i is being blamed, P consists of one or more values that constitute verifiable evidence of i's behavior, and $mess_i$ is a set of protocol messages sent by i in under the current nonce.

A proof verification function $VER \leftarrow \texttt{ProofVer}(V_i, f, P, mess_i)$ is a function run by a single party, which takes as input a public verification key V_i, a blame condition f, proof evidence P, and set of protocol messages $mess_i$. The output is True if the

proof is verified, and False if not. The protocol works as follows:

First, for each value in P, `ProofVer` checks that $mess_i$ contains a protocol message with that value included in its payload, and a current nonce and valid signature included in its metadata. If this property is not satisfied, then that value is removed from P, because it is not known that V_i actually sent the message it is being blamed for.

Once this check is complete, the exact functionality of `ProofVer` depends on the blame condition f, as described below. If P does not contain the data required for condition f, either because it was not originally included or because it was rejected in the previous step, VER is False.

- If $f = \mathsf{f}_1$, the failure is that a client or server sent an invalid key in `SetupA`. P must be a pair (K, PoK). Then $VER = \neg$ `VerifyKey`(K, PoK).

- If $f = \mathsf{f}_2$, the failure is that a server passed along an invalid client or server key in `SetupA`. P must be a pair $(\mathbf{A}, \mathbf{PoK})$. Then VER is True if `VerifyKey`(K, PoK) =False for any key in $\mathbf{A}$.

- If $f = \mathsf{f}_3$, the failure is that a client or server sent two different public keys in `SetupA`. P must be a pair (K, K'). Then VER is True if $K \neq K'$, and False else.

- If $f = \mathsf{f}_4$, the failure took place in `SetupB`, and the verification process is dependent on the implementation of the DC-nets ciphertext scheme.

- If $f = \mathsf{f}_5$, the failure took place in `VShuffle`, and the verification process is dependent on the implementation of the verifiable shuffle.

- If $f = \mathsf{f}_6$, the failure is that a client ciphertext failed to verify. P must be a tuple $(data, A, \mathbf{B}, C, PoK)$. Then $VER = \neg$ `ClientVerify`$(data, A, \mathbf{B}, C, PoK)$.

- If $f = \mathsf{f}_7$, the failure is that a server passed along a client ciphertext that failed to verify. P must be a tuple $(data, A, \mathbf{B}, C, PoK)$. Then $VER = \neg\ \texttt{ClientVerify}(data, A, \mathbf{B}, C, PoK)$.

- If $f = \mathsf{f}_8$, the failure is that a client sent two different ciphertexts. P must be a pair (C, C'). Then VER is True if $C \neq C'$, and False else.

- If $f = \mathsf{f}_9$, the failure is that a server ciphertext failed to verify. P must be a tuple $(data, \mathbf{A}, B, \mathbf{C}, S, PoK)$. Then $VER = \neg\ \texttt{ServerVerify}(data, \mathbf{A}, B, \mathbf{C}, S, PoK)$.

- If $f = \mathsf{f}_{10}$, the failure is that a server sent an invalid signature on the reconstructed message. P must be a tuple (m, N_T, σ). Then $VER = \neg\ \texttt{Verify}(V_i, (m, N_T), \sigma)$.

- If $f = \mathsf{f}_{11}$, the failure is that a server passed an invalid signature to the client at the end of a slot. P must be a tuple (V, m, N_T, σ). Then $VER = \neg\ \texttt{Verify}(V, (m, N_T), \sigma)$.

6.3 Formal Definitions of Properties

In this section we present full definitions of properties of DC-nets ciphertext schemes and verifiable shuffles, as well as properties of the full anonymity protocol. These properties, and the protocol referred to in these definitions, are based on the multi-provider cloud, anytrust architecture described in [13]. By "adversary" we mean a computationally-bounded malicious adversary. Even when these definitions specify that the adversary and challenger run certain protocols with specific inputs, the adversary is free to use any inputs it wants, and to deviate from any protocol at any time. The challenger always uses the inputs given, and follows protocols honestly

unless otherwise specified. The adversary cannot win any of these games unless the protocol completes.

6.3.1 DC-Nets Ciphertext Scheme Properties

Ciphertext scheme properties are tagged "CT-" for *ciphertext*.

CT-Anonymity

A ciphertext scheme has anonymity if a verifier cannot distinguish a client ciphertext from the anonymous slot owner's ciphertext. Formally, we say that a DC-nets ciphertext scheme has the property *CT-Anonymity* if no adversary can win at the following CT-Anonymity game with more than negligible advantage.

In the CT-Anonymity game, the adversary runs a specific protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round, during which they run a modified version of the protocol. Afterwards the adversary may run this protocol several more times before returning its output. The protocol is as follows:

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow$ `SetupA` and $(ERR, data_B, \pi) \leftarrow$ `SetupB`; or, if this is not the first round, to have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the nonce n_S incremented by one. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The adversary selects two honest clients i_0 and i_1 and a plaintext message m, and sends these to the challenger. If the adversary did not run `SetupA` and `SetupB` in the previous step, i_0 and i_1 must be the same clients the adversary used during the last round.

3. The challenger sets a bit b uniformly at random.

4. The challenger computes $(C_b, PoK_b) \leftarrow \texttt{OwnerCreate}(data, a_b, y_b, \mathbf{B}, m)$ and $(C_{1-b}, PoK_{1-b}) \leftarrow \texttt{CoverCreate}(data, a_{1-b}, \mathbf{B})$, then sends (C_0, C_1, Y_b, Y_{1-b}) to the adversary.

At the end of the challenge round, the adversary makes a guess b'. The adversary wins the game if $b = b'$.

CT-Completeness

A ciphertext scheme has completeness if an honest verifier will always accept proofs of knowledge for valid ciphertexts. Formally, we say that a DC-nets ciphertext scheme has the property *CT-Completeness* if no adversary can win at the following CT-Completeness game with probability greater than 0.

In the CT-Completeness game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may pause the challenge round at any time and run the protocol again several times with the challenger before continuing the challenge round.

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow \texttt{SetupA}$ and $(ERR, data_B, \pi) \leftarrow \texttt{SetupB}$; or, if this is not the first round, to have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the nonce n_S incremented by one. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The challenger shuffles all pseudonym public keys belonging to its clients, then sends this set of keys to the adversary.

3. The adversary replies with Y, which must be one of the keys sent in the previous step. It also sends a message m.

4. The challenger has each of its clients i run $(C_i, PoK_i) \leftarrow \texttt{CoverCreate}(data, a_i, \mathbf{B}$ except for the client whose pseudonym public key was Y; that client runs $(C_i, PoK_i) \leftarrow \texttt{OwnerCreate}(data, a_i, y, \mathbf{B}, m)$. The challenger sends (C_i, PoK_i) for each of its clients to the adversary.

5. The adversary sends the challenger a client ciphertext C_i for each of its adversarial clients.

6. The challenger collects the client ciphertexts into a set C, and has each of its servers j run $(S_j, PoK_j) \leftarrow \texttt{ServerCreate}(data, b_j, \mathbf{A}, \mathbf{C})$. It sends (S_j, PoK_j) for each of its servers to the adversary.

The adversary wins the game if the result of `ClientVerify` or `ServerVerify` returns False for any pair (C_i, PoK_i) or (S_j, PoK_j) created by the challenger during the challenge round.

CT-Integrity

A ciphertext scheme has integrity if, when at least N valid client ciphertexts and M server ciphertexts are honestly combined, the slot owner's plaintext is faithfully revealed. Formally, we say that a DC-nets ciphertext scheme has the property *CT-Integrity* if no adversary can win at the following CT-Integrity game with more than negligible probability.

In the CT-Integrity game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow$ `SetupA` and $(ERR, data_B, \pi) \leftarrow$ `SetupB`; or, if this is not the first round, to have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the nonce n_S incremented by one. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The challenger shuffles all pseudonym public keys belonging to its clients, then sends this set of keys to the adversary.

3. The adversary replies with Y, which is to serve as the slot owner's pseudonym public key. Y can be a pseudonym public key owned by one of the challenger's clients, one of the adversary's clients, or any other random piece of data.

4. The adversary sends $(data, a_i)$ to the challenger on behalf of every adversary-controlled client i. It either sends (m, y) as well on behalf of one of those clients, or it sends just m.

5. The challenger runs either `CoverCreate`$(data, a_i, \mathbf{B})$ or `OwnerCreate`$(data, a_i, y, \mathbf{B}, m)$ for each client i to get (C_i, PoK_i). The challenger runs `OwnerCreate` for the client that the adversary specified, or for the honest client for whom the challenger generated Y. If there is no such client, the adversary loses.

6. The challenger sends the set of all client ciphertexts $\mathbf{C}$ generated in the previous step to the adversary.

7. The adversary sends $(data, b_j)$ to the challenger on behalf of every adversary-controlled server j.

8. The challenger runs `ServerCreate`$(data, b_j, \mathbf{A}, \mathbf{C})$ to get (S_j, PoK_j) for each server j.

9. The challenger runs $\texttt{Reveal}(\mathbf{C}, \mathbf{S})$, where $\mathbf{S}$ is the set of server ciphertexts the challenger created, to get m'. The challenger sends m' to the adversary.

The adversary wins the game if, in the challenge round, $m \neq m'$ *and* all private keys sent by the adversary during the protocol correctly corresponded to the public keys generated for those users in the `Setup` protocols.

CT-Soundness

A ciphertext scheme is sound if an honest verifier will, with overwhelming probability, reject proofs of knowledge for invalid ciphertexts. Formally, we say that a DC-nets ciphertext scheme has the property *CT-Soundness* if no adversary can win at the following CT-Soundness game with more than negligible probability. Note that the CT-Soundness game is similar to the CT-Completeness game, except that the win condition is different; the challenger must *successfully* verify a PoK for an *invalid* ciphertext for the adversary to win.

In the CT-Soundness game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow \texttt{SetupA}$ and $(ERR, data_B, \pi) \leftarrow \texttt{SetupB}$; or, if this is not the first round, to have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the nonce n_S incremented by one. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The challenger shuffles all pseudonym public keys belonging to its clients, then sends this set of keys to the adversary.

3. The adversary declares to the challenger which type of message it wants to send, a client ciphertext or a server ciphertext.

4. If the adversary chose to send a client ciphertext:

   (a) The adversary sends Y to the challenger, which is to serve as the slot owner's pseudonym public key. Y must be one of the pseudonym public keys sent by the challenger in the previous step.

   (b) The adversary sends (C_i, PoK_i) to the challenger on behalf of one adversary-controlled client i.

   (c) The challenger runs $VER \leftarrow \texttt{ClientVerify}(data, A_i, \mathbf{B}, C_i, Y, PoK_i)$, and sends VER to the adversary.

5. If the adversary chose to send a server ciphertext:

   (a) The adversary sends Y to the challenger, which is to serve as the slot owner's pseudonym public key.

   (b) The adversary sends $(data, a_i)$ to the challenger on behalf of every adversary-controlled client i. It either sends (m, y) as well on behalf of one of those clients, or it sends just m.

   (c) The challenger runs either $\texttt{CoverCreate}(data, a_i, \mathbf{B})$ or $\texttt{OwnerCreate}(data, a_i, y, \mathbf{B}, m)$ for each client i to get (C_i, PoK_i). The challenger runs `OwnerCreate` for the client that the adversary specified, or for the honest client for whom the challenger generated Y. If there is no such client, the adversary loses.

   (d) The challenger sends the set of all ciphertexts $\mathbf{C}$ generated in the previous step to the adversary.

(e) The adversary sends (S_j, PoK_j) to the challenger on behalf of one adversary-controlled server j.

(f) The challenger runs $VER \leftarrow \texttt{ServerVerify}(data, \mathbf{A}, B_j, \mathbf{C}, S_j, PoK_j)$, and sends VER to the adversary.

The adversary wins the game if, in the challenge round, the result of `ClientVerify` or `ServerVerify` was `True` *and* the message C_i or S_j sent by the server in the challenge round was not a possible output of $\texttt{CoverCreate}(data, a_i, \mathbf{B})$ or $\texttt{ServerCreate}(data, b_j, \mathbf{A}, \mathbf{C})$, respectively, where a_i or b_j is the appropriate private key that matches the challenge public key.

CT-Zero-Knowledge

A DC-nets ciphertext scheme is zero-knowledge if a verifier learns nothing from verifying the proof of knowledge for a well-formed ciphertext besides the fact that it is well-formed. Formally, we say that a DC-nets ciphertext scheme has the property *CT-Zero-Knowledge* if there exists a polynomial-time simulator, as described below, that produces transcripts such that no adversary can win at the following CT-Zero-Knowledge distinguish game to distinguish the simulator's output from actual proofs of knowledge with more than negligible advantage.

The simulator $(PoK, \mathcal{O}') \leftarrow \texttt{Simulate}^{\mathcal{O}}(data, A_i, Y, \mathbf{B}, C_i)$ must take as input implementation-specific $data$, client public key A_i, pseudonym public key Y, the set of server public keys $\mathbf{B}$, and client ciphertext C_i. It has access to the oracle $\mathcal{O}$, either a random oracle or an oracle which returns a shared reference string, as specified by the proof of knowledge system used by the DC-nets construction. The simulator's output is a simulated proof of knowledge PoK, and a modified oracle $\mathcal{O}'$. The proof of knowledge and modification to the oracle together can be viewed as the

equivalent to the "transcript" of an interactive zero-knowledge proof of knowledge session. Our CT-Zero-Knowledge verification game will be won by the adversary only if the adversary can tell whether it is being run with this simulated "transcript" or whether it has access to the actual oracle and a genuine proof of knowledge.

In the CT-Zero-Knowledge game, the adversary runs a specific protocol with the challenger. In this game, the challenger has access to the oracle $\mathcal{O}$, but the adversary does not. The protocol is as follows:

1. All parties run $(ERR, data_A, \pi) \leftarrow$ `SetupA` and $(ERR, data_B, \pi) \leftarrow$ `SetupB`. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The challenger shuffles all pseudonym public keys belonging to its clients, then sends this set of keys to the adversary.

3. The adversary may choose to return to step 1, or to continue.

4. The adversary sends the challenger a number of tuples (Y, m, i). For each tuple, Y must be one of the keys sent by the challenger in the previous step, or the adversary loses; m is a message to be sent by the client who owns Y, and i is the honest client whose ciphertext and proof the adversary wants to see.

5. For each tuple, the challenger computes (C_i, PoK_i) by running `CoverCreate`$(data, a_i, \mathbf{B})$ for the selected client i if it was not the client for whom Y was generated, or `OwnerCreate`$(data, a_i, y, \mathbf{B}, m)$ if it was.

6. The challenger chooses a random bit b.

7. If $b = 1$, the challenger sends (C_i, PoK_i) to the adversary for each tuple it requested, and gives the adversary $\mathcal{O}$. If $b = 0$, the challenger runs `Simulate`$^{\mathcal{O}}(data, A_i, Y, \mathbf{B}, C_i)$ for each tuple to get a simulated proof PoK'_i

and oracle $\mathcal{O}'$, containing the modifications produced by all the simulations. It then sends (C_i, PoK'_i) to the adversary for each tuple it requested, and gives the adversary access to $\mathcal{O}'$).

At the end of the game, the adversary makes a guess b'. The adversary wins the game if $b = b'$.

6.3.2 Verifiable Shuffle Properties

Verifiable shuffle properties are tagged "VS-" for *verifiable shuffle*. These properties are based on those found in [8]. Note that the messages in our verifiable shuffle will be pseudonym public keys, not messages to be encrypted with DC-nets ciphertexts, but we will use the generic term "message" throughout these definitions.

VS-Anonymity

A verifiable shuffle protocol has anonymity if an adversary cannot tell which of two honest nodes submitted a given message to the shuffle. Formally, we say that a verifiable shuffle protocol has the property *VS-Anonymity* if no adversary can win at the following VS-Anonymity game with more than negligible advantage.

In the VS-Anonymity game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow$ `SetupA` and $(ERR, data_B, \pi) \leftarrow$ `SetupB`. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The adversary selects two honest clients, i_0 and i_1, and two messages m_0 and m_1, and sends these to the challenger. It also chooses messages m_i for all other honest clients i, and sends those to the challenger as well.

3. The challenger chooses a random bit b.

4. The adversary and challenger run `VShuffle` together. The challenger has each honest server j run $\texttt{ServerVShuffle}(v_j, b_j, \mathbf{B})$, and each honest client i other than i_0 and i_1 run $\texttt{ClientVShuffle}(v_i, \mathbf{B}, m_i)$. The challenger has i_0 run $\texttt{ClientVShuffle}(v_0, \mathbf{B}, m_b)$ and i_1 run $\texttt{ClientVShuffle}(v_1, \mathbf{B}, m_{1-b})$.

At the end of the game, the adversary makes a guess b'. The adversary wins the game if $b = b'$.

VS-Integrity

A verifiable shuffle protocol has integrity if, at the end of the shuffle, every honest node either receives every other honest node's message, or knows that the shuffle did not complete successfully. Formally, we say that a verifiable shuffle protocol has the property *VS-Integrity* if no adversary can win at the following VS-Integrity game with more than negligible probability.

In the P-Integrity game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to have all parties run $(ERR, data_A, \pi) \leftarrow \texttt{SetupA}$ and $(ERR, data_B, \pi) \leftarrow \texttt{SetupB}$; or, if this is not the first round, to have all parties use the same $data_A$ and $data_B$ used in the previous round, except with

the nonce n_S incremented by one. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses.

2. The adversary chooses messages m_i for all other honest clients i, and sends them to the challenger.

3. The adversary and challenger run `VShuffle` together. The challenger has each honest server j run $(ERR, [\mathbf{m}], \pi_j) \leftarrow$`ServerVShuffle`$(v_j, b_j, \mathbf{B})$, and each honest client i run $(ERR, [\mathbf{m}], \pi_i) \leftarrow$`ClientVShuffle`$(v_i, \mathbf{B}, m_i)$.

The adversary wins the game if, in the challenge round, any honest party's output from `VShuffle` has an empty proof set π *and* does not have a set $[\mathbf{m}]$ which contains each honest client's message, and one message each for all other clients.

6.3.3 Anonymity Protocol Properties

Anonymity protocol properties are tagged "P-" for *protocol*.

P-Accountability

An anonymity protocol has accountability if an honest node is able to produce a third-party verifiable proof of at least one dishonest node's behavior whenever the protocol fails, and dishonest nodes cannot produce third-party verifiable proofs blaming honest nodes. Formally, we say that a protocol has the property *P-Accountability* if no adversary can win at the following P-Accountability game with more than negligible probability.

In the P-Accountability game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to run steps 1 through 4 of `Verdict`. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, skip to step 4. If this is not the first round, the adversary may skip these steps and have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the round nonce n_R incremented by one.

2. The adversary selects plaintext messages m_i for each honest client i and sends them to the challenger.

3. The adversary and challenger run one complete round, with each client i using m_i as its message. The round ends either when the entire schedule of slots has been completed, or when a server halts.

4. At the end of the round, the adversary may submit a number of blame proofs to the challenger.

5. The challenger uses `ProofVer` to attempt to verify all proofs generated by honest parties and all proofs sent by the adversary.

At the end of the challenge round, the adversary can win in one of three ways: If `ProofVer` yielded False for any proof generated by the challenger; if a party controlled by the challenger halts before the entire schedule is completed, but does not have access to a blame proof blaming an adversarial party; or if `ProofVer` yielded True for any proof blaming an honest party.

P-Anonymity

A protocol maintains slot owner anonymity if an adversary cannot tell which client is the slot owner (assuming the slot owner is an honest client). Formally, we say that an anonymity protocol has the property *P-Anonymity* if no adversary can win

at the following P-Anonymity game with more than negligible advantage. Note that the P-Anonymity game differs from the CT-Anonymity game in that the challenger and adversary run through the entire protocol, and that the adversary may look at both ciphertexts and proofs of knowledge.

In the P-Anonymity game, the adversary runs a specific protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round, during which they run a modified version of the protocol. Afterwards the adversary may run this protocol several more times before returning its output. The protocol is as follows:

1. The adversary may choose to run steps 1 through 4 of `Verdict`. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$, the adversary loses. If this is not the first round, the adversary may skip these steps and have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the round nonce n_R incremented by one.

2. The adversary selects two honest clients i_0 and i_1 and two messages m_0 and m_1 and sends these to the challenger.

3. The challenger selects a random bit b.

4. The adversary and challenger run one complete round, with client i_0 using m_b as its message, and client i_1 using message m_{1-b}. The round ends either when the entire schedule of slots has been completed, or when a server halts.

At the end of the challenge round, the adversary makes a guess b'. The adversary wins the game if $b = b'$.

P-Integrity

An anonymity protocol has integrity if, when a protocol round finishes, every honest client either obtains the slot holder's message or knows that the protocol did not complete successfully. Formally, we say that an anonymity protocol has the property *P-Integrity* if no adversary can win at the following P-Integrity game with more than negligible probability. (Note that the P-Integrity game differs from the CT-Integrity game in that the adversary is allowed to produce invalid ciphertexts, as long as this is not detected.)

In the P-Integrity game, the adversary runs a protocol with the challenger. They run this protocol several times, until the adversary signals that it is ready to begin the challenge round. The adversary may run the protocol again several times with the challenger before sending its final message of the challenge round.

1. The adversary may choose to run steps 1 through 4 of `Verdict`. If any server controlled by the challenger gets $ERR = \mathsf{FAIL}$ during any of these steps, the adversary loses. If this is not the first round, the adversary may skip these steps and have all parties use the same $data_A$ and $data_B$ used in the previous round, except with the round nonce n_R incremented by one.

2. The adversary selects plaintext messages m_i for each honest client i and sends them to the challenger.

3. The adversary and challenger run one complete round, with each client i using m_i as its message. The round ends either when the entire schedule of slots has been completed, or when a server halts.

The adversary wins the game if, in the challenge round, the output of each honest client does not contains every message m_i for that round and contains no blame

proofs.

6.4 Proofs of Protocol Properties to DC-Nets Ciphertext Properties

Here are proofs showing that if the protocol has the CT properties, then it will have the P properties as well.

Terminology

By "valid ciphertext", we mean any ciphertext in the output space of the `Create` methods of the DC-nets ciphertext system, given proper inputs.

That is, a valid ciphertext from non-owner client i would be any output of `CoverCreate`$(data, a_i, \mathbf{B})$, where *data* is the appropriate implementation-dependent data, a_i is the private key corresponding to the public key A_i that i shared in `SetupA`, and $\mathbf{B}$ is the set of server public keys shared during `SetupA`. If i is the slot owner, a valid ciphertext is any output of `OwnerCreate`$(data, a_i, y, \mathbf{B}, m)$, where *data*, a_i, and $\mathbf{B}$ are the same as before; y is the pseudonym private key corresponding ot this slot's pseudonym public key, and m can be anything.

A valid ciphertext from server j is any output of `ServerCreate`$(data, b_j, \mathbf{A}, \mathbf{C})$, where *data* is the appropriate implementation-dependent data, b_j is the private key corresponding to the public key B_j that j shared in `SetupA`, $\mathbf{A}$ is the set of client public keys shared during `SetupA`, and $\mathbf{C}$ is the set of client ciphertexts in this slot.

6.4.1 Proof of P-Accountability

Theorem 3. *If* `Verdict` *uses a DC-nets ciphertext system with the properties of CT-Completeness, CT-Integrity, and CT-Soundness, then* `Verdict` *has the property of P-Accountability.*

Proof. Recall from section 6.3.3 that there are three cases in which P-Accountability is compromised:

(i) A party following the protocol has a proof in its output for which `ProofVer` returns False

(ii) A party following the protocol halts without having any proof in its output

(iii) An adversary can produce a fraudulent proof blaming an honest party for which `ProofVer` returns True

An inspection of the `Verdict` protocol makes it clear that case ii will not occur. In every case where a party following the protocol halts, it first adds a proof to its output. It therefore remains to show that for each failure condition f, `ProofVer` will return True if and only if its input proof was constructed by a party following the protocol honestly. To demonstrate this, we will investigate every failure condition one by one.

We rely on the unforgeability property of the signature scheme to guarantee that all messages being used to blame a user were genuinely sent by that user. In the following examination, we will also rely on the following claim:

- f_1 **Invalid key proof** We rely on the existence of a complete and sound non-interactive zero-knowledge proof-of-knowledge system for keypairs in the chosen DC-nets ciphertext system. As long as such a NIZKPoK system exists, by the

completeness of that system `VerifyKey` will always correctly verify an honest user's proof as generated by `KeyGen`, so proofs of this type cannot be forged. Furthermore, `VerifyKey` will, except with negligible probability, fail to verify a proof of a user who does not know the secret key k it purports to know, so the proof generated by an honest user will be verified by `ProofVer`.

- f_2 **Forwarded invalid key** As above, valid proofs will be verified by `VerifyKey`, and invalid proofs will not. In this case, the forwarding server's signature on the message-proof pair certifies its guilt.

- f_3 **Key equivocation** If a party sends two different public keys in `SetupA`, the difference will be obvious, so an honest party's proof will consist of the two keys. Because all messages are signed using a nonce and step number, only if a party signs two different keys during steps 2 or 5 of `SetupA` can this kind of proof be created. No party honestly following the protocol will do that.

- f_4 `SetupB` **failure** The verification of this failure condition depends on the implementation of `SetupB` in the ciphertext scheme.

- f_5 `VShuffle` **failure** The verification of this failure condition depends on the implementation of `VShuffle`.

- f_6 **Invalid client ciphertext** Due to CT-Completeness, there can be no honestly-generated input $(data, a_i, \mathbf{B})$ to `CoverCreate` or $(data, a_i, y, \mathbf{B}, m)$ to `OwnerCreate` that will cause the algorithm to return a pair (C_i, PoK_i) that will not verify with `ClientVerify`. If there were any such input, the adversary would have a non-zero probability of winning the CT-Completeness game. Therefore, a blame proof using the tuple $(data, A_i, \mathbf{B}, C_i, Y, PoK_i)$ cannot be used to blame an honest client, because if an honest client generated it then

`ClientVerify` will return True. By the same token, if `ClientVerify` returns False, we can be sure that the input included in the blame proof was actually generated by a dishonest client, and `ClientVerify` will return False on it. So an honest server will reliably be able to construct this kind of proof.

- f_7 **Forwarded invalid ciphertext** As argued above, if `ClientVerify` returns False, its input must have been created by a dishonest client. Furthermore, since `ClientVerify` is deterministic, if `ClientVerify` returns False once, it will always return False on that same input. An honest server following the protocol should have removed a ciphertext which failed to verify, and a message from it containing an unverified ciphertext is proof of misbehavior which other honest parties will recognize. No honest server would have sent such a message, so an honest server cannot be blamed using this method.

- f_8 **Client equivocation** If a client sends two different valid ciphertexts to two servers in step 5(a)iv, the difference between the two ciphertexts will be obvious. Again, the metadata will ensure that a proof of this kind cannot be forged.

- f_9 **Invalid server ciphertext** Due to CT-Completeness, there can be no honestly-generated input $(data, \mathbf{A}, B, \mathbf{C}, S, PoK)$ to `ServerCreate` that will cause the algorithm to return a pair (S_j, PoK_j) that will not verify with `ServerVerify`. This follows precisely the same logic as in the case of an invalid client ciphertext, above.

- f_{10} **Invalid signature** An additional signature is required from each server to verify the reconstructed message to the clients. All honest servers will, if they reach this point in the protocol, have the same set of client ciphertexts $\mathbf{C}$ and

a set of valid server ciphertexts **S**. All honest servers should therefore recover the same message m and produce the same signature. If there were a set of keys that would have allowed different honest servers to recover the different ciphertexts in step 5(a)xiii of `Verdict`, then an adversary to the CT-Integrity game could use those keys to win that game. Therefore, there is at most a negligible probability that an honest server can be blamed this way. Except in that unlikely case, proofs of this type will only happen if dishonest servers create false signatures, which honest servers will all be able to recognize by the soundness of the signature scheme.

- f_{11} **Forwarded invalid signature** Any server following the protocol will not forward invalid signatures from other servers, so this type of proof can only be used to blame dishonest servers. Honest clients will always be able to construct this kind of proof as long as the signature scheme is complete and sound.

We have shown how, for each failure which can halt the `Verdict` protocol, an honest node can generate third-party verifiable proof of misbehavior, and that these proofs cannot be used to blame honest parties. Therefore, the protocol has the property of P-Accountability. □

6.4.2 Proof of P-Anonymity

Theorem 4. *If `Verdict` uses a DC-nets ciphertext system with the properties of CT-Anonymity and CT-Zero-Knowledge, and a verifiable shuffle `VShuffle` with VS-Anonymity, then `Verdict` has the property of P-Anonymity.*

Proof. We will use a hybrid argument to show that the advantage of adversary A in the P-Anonymity game is negligible. The strategy we are using is to replace each step of the protocol with an ideal or simulated version, one at a time, in each step

proving that the adversary's output distribution will be at most negligibly different from before. Finally, we will end with a game in which we will directly prove that the adversary's advantage is negligible.

- Game 0 is the P-Anonymity game.
- Game 1 is modified from game 0 in that the challenger swaps the inputs of the two challenge clients to `ClientVShuffle`, having challenge client 0 submit Y_1 as its message, and challenge client 1 submit Y_0 as its message.
- Game 2 is modified from Game 1 in that the entire `VShuffle` protocol is replaced with a random permutation performed by the challenger.
- Game 3 is modified from Game 2 in that the all honest clients use simulated proofs of knowledge rather than real ones.

We will now use a series of lemmas to show that the output distribution varies no more than negligibly from one game to the next.

Lemma 5. *The probability that the adversary A wins Game 0 differs from the probability that A wins Game 1 by no more than a negligible amount.*

Proof. Suppose the opposite, that A's output distribution differs between Games 0 and 1 by a non-negligible amount. Then we can construct an adversary B for the VS-Anonymity game which will win with more than negligible advantage, contradicting the assumption that `VShuffle` offers VS-Anonymity. B will use the A as an oracle, running Game 0 with it and acting as the challenger.

B would have two honest clients to distinguish between in the P-Anonymity game, i_0 and i_1. So B would choose two corresponding honest users i_0 and i_1 in its VS-Anonymity game, and use those two clients' pseudonym public keys, Y_0 and Y_1,

as its challenge messages. In the shuffle step, B would transmit all messages from the VS-Anonymity challenger to A and vice versa. It would then follow the protocol to completion with A. B's output in the VS-Anonymity game would be the same as A's output.

During the shuffle step, the VS-Anonymity challenger may have had i_0 submit PK_0 and i_1 submit PK_1, or it may have had i_0 submit PK_1 and i_1 submit PK_0. If we fix a challenge bit in B's game with A, we can see that B was playing Game 0 with A in the former case, and Game 1 in the latter case. Since B is just copying A's output, any non-negligible difference in output distribution between Game 0 and Game 1 would mean that B also has a non-negligible advantage distinguishing between the two cases in the VS-Anonymity game. This would contradict our assumption that the verifiable shuffle in fact has anonymity. Therefore, there can be no negligible difference in output distribution between Game 0 and Game 1. □

Lemma 6. *The probability that the adversary A wins Game 1 differs from the probability that A wins Game 2 by no more than a negligible amount.*

Proof. Game 2 has the same output distribution as Game 0 with 50% probability, and the same output distribution as Game 1 with 50% probability. By the preceding lemma, these two output distributions are negligibly close to each other. Therefore, Game 2's output distribution is also negligibly close to that of Game 1. □

Lemma 7. *The probability that the adversary A wins Game 2 differs from the probability that A wins Game 3 by no more than a negligible amount.*

Proof. Suppose the opposite, that A's output distribution differs between Games 2 and 3 by a non-negligible amount. Then we can construct an adversary B for the CT-Zero-Knowledge game which will win with more than negligible advantage, contradicting the assumption that the ciphertext system offers CT-Zero-Knowledge.

B will use A as an oracle, acting as the challenger in the P-Anonymity game, except replacing the `VShuffle` protocol with a permutation as in Game 2. The operation of B would then be as follows:

1. B transmits messages between its own challenger and A without modification in order to complete the first step of the P-Anonymity game, repeating step 1 of the CT-Zero-Knowledge game as necessary to provide responses.

2. B uses the clients and messages that A chooses in step 2 of the P-Anonymity game to generate request tuples in step 4 of the CT-Zero-Knowledge game. It generates one set of tuples for each of the honest clients' public pseudonym keys. In other words, B requests from the CT-Zero-Knowledge challenger enough ciphertext/proof pairs for its clients to be able to participate in one complete round of the P-Anonymity game.

3. The CT-Zero-Knowledge challenger will, in step 7, return the requested pairs (C, PoK). The proofs may be real or simulated. The challenger also gives B access to an oracle $\mathcal{O}$ which may have been modified; B gives A access to the same oracle.

4. In the P-Anonymity game, B fixes a challenge bit b, and continues running the game with A. It has each of its clients i send the appropriate pair (C_i, PoK_i) in each slot.

5. If A wins its game, B outputs 1, signifying real proofs of knowledge. Otherwise, B outputs 0, signifying simulated proofs of knowledge.

We can see that B was playing Game 2 if it had actual proofs of knowledge, and Game 3 if it had simulated proofs. If we fix a challenge bit in B's game with A, B is either always copying A's output or always returning the opposite of A's output;

in either case, any non-negligible difference in output distribution between Game 2 and Game 3 would mean that B also has a non-negligible distinguishing between real and simulated proofs in the CT-Zero-Knowledge game. This would contradict our assumption that the DC-nets ciphertext system in fact has CT-Zero-Knowledge. Therefore, there can be no negligible difference in output distribution between Game 2 and Game 3. □

Lemma 8. *If the DC-nets ciphertext system has CT-Anonymity, A's advantage in winning Game 3 is negligible.*

Proof. Suppose the opposite, that A's advantage in Game 3 is non-negligible. Then we can construct an adversary B for the CT-Anonymity game which will win with more than negligible advantage, contradicting the assumption that the ciphertext system offers CT-Anonymity. B will use A as an oracle, acting as the challenger in the P-Anonymity game, except replacing the `VShuffle` protocol with a permutation as in Game 2 and replacing all proofs of knowledge with simulations as in Game 3. The operation of B would then be as follows:

1. B transmits messages between its own challenger and A without modification in order to complete the first step of the P-Anonymity game up to the part where `VShuffle` would take place.

2. B runs the CT-Anonymity protocol through step 4 in order to get pseudonym public keys and one pair of ciphertexts for the challenge clients.

3. B permutes the pseudonym public keys randomly, and sends the permuted list to A. It records the slots belonging to the two challenge clients, but it does not know which slot goes with which client.

4. B runs the CT-Anonymity protocol again through step 4, until it gets a pair of ciphertexts corresponding to the other challenge client's slot. (In other words, it repeats the protocol until the two pseudonym public keys returned in step 4 are in the other order.)

5. Now, B has learned from the CT-Anonymity challenger two pairs of client ciphertexts, C_0^b, C_0^{1-b}, C_1^b, and C_1^{1-b}, and two pseudonym public keys, Y_b and Y_{1-b}, where C_i^b is client i's ciphertext in the slot where Y_b is pseudonym public key of the slot owner. b is the challenge bit in the CT-Anonymity game, which B does not know. So B temporarily presumes that $b = 0$. It randomly chooses its own challenge bit $\bar{b}$ for its game with A.

6. B continues running the P-Anonymity game with A, having its challenge clients send the appropriate ciphertexts in step 5(a)iv. For example, when i_b is the slot holder, client i_0 sends ciphertext C_0^b if B chose $\bar{b} = 0$, or ciphertext C_1^b if it chose $\bar{b} = 1$. It simulates proofs of knowledge as in Game 3.

7. At the end of the game, if A wins, B returns 0. If A loses, B returns 1.

If B's presumption that $b = 0$ is correct, A and B are exactly playing Game 3 with challenge bit $\bar{b}$. If B's presumption was incorrect, A and B are playing Game 3 except that the ciphertexts submitted by i_0 and i_1 have been switched *twice*. In other words, A and B were playing Game 3, but with challenge bit $1 - \bar{b}$, and A's output distribution is exactly reversed, so there is now a greater-than-negligible chance that A *loses* the game. So, by returning 0 when A wins and 1 when A loses, in either case B has a greater-than-negligible chance of winning the CT-Anonymity game. This contradicts our CT-Anonymity assumption, so no such A can exist. □

We have shown that, as long as CT-Zero-Knowledge and VS-Anonymity hold,

any adversary A has only a negligible advantage over any CT-Anonymity adversary, and by assumption, no CT-Anonymity adversary has more than a negligible advantage itself. Therefore, no adversary in the P-Anonymity game has a non-negligible advantage. □

6.4.3 Proof of P-Integrity

Theorem 5. *If* `Verdict` *uses a DC-nets ciphertext system with the properties of CT-Soundness and CT-Integrity, then* `Verdict` *has the property of P-Integrity.*

Proof. For P-Integrity to be violated, there must be a non-negligible chance that, at the end of a round, the plaintext messages revealed are not the ones submitted by the clients and yet no honest parties have any proofs of misbehavior. Assume there is an adversary A which can win the P-Integrity game with non-negligible probability.

There are two cases: A wins by having its clients and servers send only valid ciphertexts in all steps, or A wins by having one of its clients or servers send an invalid ciphertext at some point. We will argue that the existence of A violates CT-Integrity in the former case and CT-Soundness in the latter case.

Consider the restricted case of the P-Integrity game where A sends only valid ciphertexts. It is clear that each round of this restricted P-Integrity game is identical to a round of the CT-Integrity game, except that A is allowed to choose which valid ciphertext will be sent for each of its clients and servers. But if there is any ciphertext which could cause the message not to be reconstructed, then the adversary to the CT-Integrity game has a positive probability of winning (when the challenger selects this ciphertext). This would violate the assumption that the DC-nets ciphertext system provides CT-Integrity.

In order to win the P-Integrity game with non-negligible probability, A must

therefore send an invalid ciphertext at some step. However, A loses if a proof of its misbehavior is created. As demonstrated in the proof of P-Accountability above, if an invalid ciphertext is detected, a proof will be created. So all it remains to show is that invalid ciphertexts will be detected with overwhelming probability.

This we can do by reduction to CT-Soundness. Any adversary A that can produce a ciphertext/proof pair which verifies with `ClientVerify`, but which does not actually have a valid ciphertext, could use that same pair to win at the CT-Soundness game. Therefore, as long as the ciphertext system provides CT-Soundness, no such A can exist. □

Chapter 7

Conclusions

7.1 Openness in Lawful Surveillance

In Chapter 2, we proposed a system of principles that might govern lawful surveillance. The principles are a first stab at an appropriate foundation for privacy-preserving, accountable surveillance. We hope that they contribute to a new and growing dialog about how cryptographic technology can allow surveillance and privacy to co-exist, and that dialogue will stimulate further discussion and continue to be refined and revised by the relevant research communities.

One remaining high-level issue is the tension between the openness principle proposed in Section 2.1 – requiring that processes handling "bulk" electronic surveillance data be open – and the traditional desire of intelligence agencies to protect "sources *and methods*," especially from the knowledge of criminals or terrorists being investigated. We emphasize that satisfying our openness principle by no means demands exposing *all* intelligence methods – only those few involved in implementing the "privacy firewall" in Figure 2.1b. All the details of any particular investigation – who is being investigated, when, the details of a particular warrant such as which metadata

sets are to be intersected, how those sets were chosen, and how the decrypted results are processed only after being lawfully queried through the "privacy firewall" – could still rely on closely guarded intelligence methods.

A generic open process such as the set-intersection primitive tends to be usable in many different, specific ways – as in the contrasting High Country Bandits [1] and NSA CO-TRAVELER [49] examples. Had CO-TRAVELER not been disclosed by Snowden, for example, then this specific method of using set intersection to find unknown associates of known targets as they travel might well remain a closely guarded secret, even if the basic intersection-warrant mechanism were well-known, openly debated, and instituted in public policy.

Finally, a basic tenet of democratic society and the rule of law is that it is better to risk a few criminals' going uncaught, because they know and understand public law-enforcement processes "too well," than to risk that secret law-enforcement processes, however well intentioned at the outset, might become unaccountable and evolve into "star-chamber" tools of political repression and authoritarianism. This democratic principle of openness must be carried into the electronic world; with the right tools, the principle need not tie the hands of legitimate, accountable law-enforcement processes.

7.2 Privacy-Presering Set Intersection

From the experimental results in Section 3.2, we conclude that privacy-preserving, accountable set intersection may indeed be achievable at scale. This in turn leads us to be optimistic about the feasibility of the broader goal articulated in Chapter 2: maintaining constitutional rights in powerful, evolving, digital-communication systems while simultaneously equipping law-enforcement and intelligence agencies to

use these systems to combat and prevent crime and terrorism. There is a great deal of further work to be done along these lines, and we briefly describe a portion of it here.

7.2.1 Enhancements and Generalizations

The privacy-preserving set intersection protocol that we described in Section 3.1.2 leaks the sizes of pairwise intersections (but not the contents of those intersections) in the case of three participants; more generally, it leaks the sizes of the j-wise intersections, where $1 < j < k$, and k is the number of participants. As explained in Sections 3.1 and 3.2, this is not a show stopper on privacy or efficiency grounds, but it leaves open the question of whether there is a similarly efficient protocol with the same accountability properties that reveals no information except the k-wise intersection.

In principle, one could achieve this ideal level of privacy by starting with a general secure, multiparty computation (SMPC) protocol and augmenting it with the appropriate accountability features. How well such an approach would scale is an open question.

Starting with the Fairplay platform for secure, two-party computation [41], there has been much work on general-purpose SMPC platforms. The goal of this research is to provide languages, compilers, run-time environments, and other platform elements that enable programmers who are not experts in cryptography or SMPC to write ordinary code and transform it into executable, multiparty protocols with the desired security properties. There are now many such platforms whose performance and usability are improving (see, *e.g.,* [60]). One could, in principle, achieve the goals put forth in Section 3.1 simply by writing a set-intersection program and using, say, Sharemind [7] to translate it into a privacy-preserving, distributed set-

intersection protocol (rather than implementing privacy-preserving set intersection "from scratch" as in Section 3.2). Open questions include whether the resulting protocol would be efficient enough to use at scale and how to make it accountable.

7.3 Privacy-Preserving Contact Chaining

For contact chaining, it may be possible to speed up our protocols by using elliptic-curve cryptography instead of RSA. Additionally, our assumption that all parties behave in an honest-but-curious manner might be weakened. By using standard zero-knowledge proof techniques, it might be possible to create versions of the protocols in Section 4.1 that are secure against, for example, a rogue agent's maliciously modifying telecom-supplied data in order to falsely incriminate a victim. It may also be interesting to generalize the differential-privacy approach of Kearns *et al.* [38] so that it applies to indirect contacts as well as direct contacts.

Of course, intersection of cell-tower dumps and contact-chaining are just two of many computations that could be of use to law-enforcement and intelligence agencies. It would be interesting to identify other such computations and to apply to them the principles and computational approaches that we have explored.

Another problem of potential interest is the retrieval of targeted users' postings on Facebook and other social networks, including those that are shared only with a small subset of the targeted user's "friends." Accountable surveillance of social-network postings may present novel protocol-design challenges, because it deals with one-to-many communication, whereas previous work in the area dealt with pairwise communication.

7.4 PeerFlow

Tor's security is vulnerable to an adversary running large relays, and we show that under the Tor's current TorFlow bandwidth-measurement system and the proposed EigenSpeed system an adversary can make small relays appear large, drastically reducing the cost of attack. We present PeerFlow, show how it limits the ability of an adversary to fool Tor about the bandwidth of his relays, and demonstrate that its performance is comparable to Tor's currently. Possible future improvements to PeerFlow include improving scalability and further improving security to reduce the adversary's ability to inflate his relays size even more.

7.5 Final Thoughts

Government authorities need wait for no "magic bullet" breakthrough in cryptographic technology, nor use intentionally weakened encryption schemes. As we have shown, privacy-preserving surveillance would allow them to legally acquire actionable and relevant information about the targets of their investigation and still protect the privacy of innocent people. From at least a technological perspective, privacy-preserving surveillance could start being deployed immediately.

Similarly, internet users can protect their identity online with Tor, which is already available for download. Tor is under continuous development, and we hope that our PeerFlow system will soon replace the exploitable and insecure TorFlow as a tool for bandwidth measurement. We also hope that our proof of accountable anonymity for Verdict spurs further research and development into DC-nets-based anonymity, which ideally could be deployed alongside Tor as an alternative for people with an absolute need for anonymous communication.

By showing how the bulk surveillance process can be opened up to privacy-

preserving mechanisms, and how anonymous communication systems can be made secure in both practical and theoretical environments, we have demonstrated that there is no need to accept the total loss of control over their own data that most users seem to face. Both governmental authorities and ordinary people can take steps to adopt these technologies instead, and make private information truly private once again.

Bibliography

[1] Nate Anderson. How "cell tower dumps" caught the High Country Bandits—and why it matters. *arstechnica*, August 29, 2013.

[2] Bandwidth scanner spec. `https://gitweb.torproject.org/torflow.git/blob_plain/HEAD:/NetworkScanners/BwAuthority/README.spec.txt`.

[3] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proc. of ACM Symposium on Principles of Distributed Computing*, 2001.

[4] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In *Proc. of ACM Workshop on Privacy in the Electronic Society*, 2007.

[5] Meredith A. Bieber. Meeting the statute or beating it: Using John Doe indictments based on DNA to meet the statute of limitations. *University of Pennsylvania Law Review*, 150(3):1079–1098, 2002.

[6] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, measurement, deanonymization. In *Proc. of IEEE Symposium on Security and Privacy*, 2013.

[7] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[8] Justin Brickell and Vitaly Shmatikov. Efficient anonymity-preserving data collection. In *Proc. of ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2006.

[9] Claude Castelluccia, Mohamed-Ali Kaafar, and Minh-Dung Tran. Betrayed by your ads! reconstructing user profiles from targeted ads. In *Proc. of Privacy Enhancing Technologies Symposium*, 2014.

[10] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, pages 65–75, January 1988.

[11] CollecTor. `https://collector.torproject.org/`.

[12] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable anonymous group messaging. In *Proc. of ACM Conference on Computer and Communications Security*, October 2010.

[13] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in Verdict. In *Proc. of USENIX Security*, August 2013.

[14] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Proc. of CRYPTO*, 1994.

[15] Tim Cushing. NSA Appears To Be Chaining Calls Using Phone Numbers One Hop Out As New Originating Selectors. *Techdirt*, July 3, 2014.

[16] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Proc. of IACR International Conference on Practice and Theory of Public-Key Cryptography*, 2001.

[17] Ivan Damgård and Mads Jurik. A length-flexible threshold cryptosystem with applications. In *Proc. of Australasian Conference on Information Security and Privacy*, 2003.

[18] Ryan Devereaux, Glenn Greenwald, and Laura Poitras. Data Pirates of the Caribbean: The NSA Is Recording Every Cell Phone Call in the Bahamas. *The Intercept*, May 20, 2014.

[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proc. of USENIX Security*, 2004.

[20] Cynthia Dwork. Differential privacy. In *Proc. of International Colloquium on Automata, Languages and Programming*, 2006.

[21] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[22] Nathan Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on Tor using long paths. In *Proc. of USENIX Security*, 2009.

[23] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *Proc. of Financial Cryptography*, 2000.

[24] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proc. of EUROCRYPT*, 2004.

[25] Philippe Golle and Ari Juels. Dining cryptographers revisited. *Proc. of EUROCRYPT*, May 2004.

[26] Glenn Greenwald. NSA collecting phone records of millions of Verizon customers daily. *The Guardian*, June 6, 2013.

[27] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proc. of ACM Symposium on Operating Systems Principles*, 2007.

[28] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security*, 13(2), February 2010.

[29] Human Rights Council. The right to privacy in the digital age: Report of the Office of the United Nations High Commissioner for Human Rights, June 2014.

[30] Tom N. Jagatic, Nathaniel A. Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, October 2007.

[31] Rob Jansen, Kevin Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the Tor network. In *Proc. of USENIX Workshop on Cyber Security Experimentation and Test*, 2012.

[32] Rob Jansen, John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson. Never been KIST: Tor's congestion management blossoms with kernel-informed socket transport. In *Proc. of USENIX Security*, 2014.

[33] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a box for accurate and efficient experimentation. In *Proc. of ISOC Network and Distributed System Security Symposium*, 2012.

[34] Rob Jansen, Aaron Johnson, and Paul Syverson. LIRA: Lightweight incentivized routing for anonymity. In *Proc. of ISOC Network and Distributed System Security Symposium*, 2013.

[35] Rob Jansen, Andrew Miller, Paul Syverson, and Bryan Ford. From onions to shallots: Rewarding Tor relays with TEARS. In *Proc. of Workshop on Hot Topics in Privacy Enhancing Technologies*, 2014.

[36] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proc. of ACM Conference on Computer and Communications Security*, 2013.

[37] Ghassan Karame, David Gubler, and Srdjan Capkun. On the security of bottleneck bandwidth estimation techniques. In *Proc. of EAI International Conference on Security and Privacy in Communication Networks*, 2009.

[38] Michael Kearns, Aaron Roth, Zhiwei Steven Wu, and Grigory Yaroslavtsev. Private algorithms for the protected in social network search. *Proceedings of the National Academy of Sciences*, 113(4):913–918, 2016.

[39] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Proc. of CRYPTO*, 2005.

[40] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[41] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay – a secure two-party computation system. In *Proc. of USENIX Security*, August 2004.

[42] mobiThinking. Global mobile statistics 2014 Part A: Mobile subscribers; handset market share; mobile operators. *mobiForge*, May 16, 2014.

[43] Mike Perry. Torflow: Tor network analysis. In *Proc. of Workshop on Hot Topics in Privacy Enhancing Technologies*, 2009.

[44] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[45] Aaron Segal, Bryan Ford, and Joan Feigenbaum. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, August 2014. USENIX Association.

[46] Robin Snader. *Path Selection for Performance- and Security-Improved Onion Routing*. PhD thesis, U. of I. at Urbana-Champaign, 2009.

[47] Robin Snader and Nikita Borisov. Eigenspeed: Secure peer-to-peer bandwidth evaluation. In *Proc. of International Workshop on Peer-to-Peer Systems*, 2009.

[48] Robin Snader and Nikita Borisov. Improving security and performance in the Tor network through tunable path selection. *IEEE Transactions on Dependable and Secure Computing*, 8(5):728–741, September 2011.

[49] Ashkan Soltani and Barton Gellman. New documents show how the NSA infers relationships based on mobile location data. *The Washington Post*, December 10, 2013.

[50] R. Suselbeck, G. Schiele, P. Komarnicki, and C. Becker. Efficient bandwidth estimation for peer-to-peer systems. In *Proc. of IEEE International Conference on Peer-to-Peer Computing*, 2011.

[51] Fabrice Thill. *Hidden Service Tracking Detection and Bandwidth Cheating in Tor Anonymity Network*. PhD thesis, Univ. Luxembourg, 2014.

[52] Tor directory protocol, version 3. `https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=dir-spec.txt`.

[53] Tor metrics. `https://metrics.torproject.org/`.

[54] Yiannis Tsiounis and Moti Yung. On the security of ElGamal based encryption. In *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 1998.

[55] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.

[56] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proc. of USENIX Security*, 2014.

[57] Tao Wang and Ian Goldberg. Improved website fingerprinting on Tor. In *Proc. of ACM Workshop on Privacy in the Electronic Society*, 2013.

[58] Philipp Winter, Richard Kwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious Tor exit relays. In *Proc. of Privacy Enhancing Technologies Symposium*, 2014.

[59] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Sys-

tems. *ACM Transactions on Information and System Security*, 4(7):489–522, November 2004.

[60] Yihua Zhang, Aaron Steele, and Marina Blanton. Picco: A general-purpose compiler for private distributed computation. In *Proc. of ACM Conference on Computer and Communications Security*, 2013.