PROTECTING PRIVACY BY SPLITTING TRUST

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Henry Corrigan-Gibbs
December 2019

Abstract

In this dissertation, we construct two systems that protect privacy by *splitting trust* among multiple parties, so that the failure of any one, whether benign or malicious, does not cause a catastrophic privacy failure for the system as a whole. The first system, called Prio, allows a company to collect aggregate statistical data about its users without learning any individual user's personal information. The second, called Riposte, is a system for metadata-hiding communication that allows its users to communicate over an insecure network without revealing who is sending messages to whom. Both systems defend against malicious behavior using *zero-knowledge proofs on distributed data*, a cryptographic tool that we develop from a new type of probabilistically checkable proof.

The two systems that we construct maintain their security properties in the face of an attacker who can control the entire network, an unlimited number of participating users, and any proper subset of the servers that comprise the system. These systems split trust in the sense that, as long as an attacker cannot compromise *all* of the participating servers, the system provides "best-possible" protection of the confidentiality of user data. Through the design, implementation, and deployment of these systems, we show that it is possible for us to enjoy the benefits of modern computing while protecting the privacy of our data.

To my teachers

Acknowledgments

My cousin Joe often jokes that I am a lifetime student. It occurred to me recently that this isn't really a joke. So, as a lifetime student, I want to dedicate this dissertation to my teachers—all of the people in my life who have shared their knowledge and experience with me and who have enriched my life in the process.

The Applied Crypto Group has been my academic home on campus and most of what I learned in grad school, I learned from its tremendous set of students and postdocs: Alex Ozdemir, Ananth Raghunathan, Ben Fisch, Benedikt Bünz, Benton Case, David Wu, Dmitry "Dima" Kogan, Florian Tramèr, Giancarlo Pellegrino, Hart Montgomery, Joe Bonneau, Joe Zimmerman, Kevin Lewi, Mark Zhandry, Riad Wahby, Saba Eskandarian, Sam Kim, Sergio Benitez, Suman Jana, Valeria Nikolaenko, and Yan Michalevsky.

In my first year at Stanford, I shared an office with the members of the Secure Computer Systems Group. Those students, including Adam Belay, Ali Mashtizadeh, Amit Levy, Daniel Giffin, David Terei, Deian Stefan, and Edward Z. Yang, taught me volumes about programming, systems, and computer security. I want to thank Edward especially for many enlightening bike rides in the foothills and for his sage advice at a number of points in grad school when I needed it most.

Later in grad school, I had the pleasure of working in an office with Francis Y. Yan, Greg Hill, John Emmons, Riad S. Wahby, and Sadjad Fouladi. I benefited immensely from Greg's good taste in all things—from research questions and chart design to Bay Area restaurants and office decor. Talking with John about technical questions, especially on our Saturday-morning rides on Skyline, was terrific. The dedication and precision that Riad brings to his work consistently blows me away; I thank him for being so generous in giving feedback on my talks, papers, and half-baked ideas throughout my time at Stanford. And I thank Sadjad for being my source of truth on programming in general, and C++ in particular.

During the latter part of grad school, I worked closely with a set of student co-authors and with my co-teachers of CS355 and CS359C: Sam Kim, Emma Dauterman, Florian Tramèr, Saba Eskandarian, Albert Kwon, David Wu, and Dima Kogan. Sam, you rarely raise your hand to ask a question or make a comment in talks, but when you do, without fail, you make an insightful observation that the rest of us would have missed. Emma, I continue to be impressed by the level of dedication and drive that you bring to your work and I look forward to seeing what you accomplish in this next phase of your research career. Florian, I greatly enjoyed our philosophical discussions about computer security (and your dark sense of humor about it all) at lunchtime, in the reading groups, and—most of all—on the ski lift. Saba, I feel very fortunate to have had you as a colleague and friend for these years, to share in both the successes of grad school and to commiserate when things didn't go to plan. It's hard to imagine working in an office without you around the corner. Albert, it was great fun working with you—you manage to be extremely focused and hardworking while also being very cheerful and laid back. It has been so satisfying to see your efforts in grad school pay off. David, you are one of my research role models. Even as your list of accomplishments and responsibilities grows longer and longer, you continue to maintain the same kindness, patience, and humility that you have always had. It is reassuring in this post-PhD life to know that if I ever get really stuck on something, I can always give you a call. Dima, discussing research problems with you in the various cafés around campus was a highlight of my time at Stanford. I learned so much about research just by watching you think and write, and I hope to have many more chances to work with you in the future.

The staff members in the Gates Building were outstanding. I can't thank them enough. Jay Subramanian expertly handled all sorts of issues that came up during my PhD studies. I never went wrong by following her advice. Angela Cao and Jam Kiattinant made sure that my funding was always in the right place at the right time. Mary Jane Swenson was a problem-solver *extraordinaire*. Even though she worked in a different group and on a different floor, she was an unfailingly reliable source of advice and help. Megan Harris and Ruth Harris were indispensable. They made sure that I always had the resources I needed to be a happy and productive grad student. Room bookings, travel funding, catering, supplies, office space, and on and on—they took care of it all with such class and such professionalism. I am in awe.

I have worked with an excellent set of research external collaborators. This group includes Wendy Mu, Jay Chen, Bill Thies, Ed Cutrell, Nakull Gupta, Curtis Northcutt, Judson Wilson, Stuart Schechter, Srini Devadas, Eric Rescorla, Robert Helmer, Anthony Miyaguchi,

and Dominic Rizzo. I thank them for exposing me to new areas of computer science and for teaching me to be a better researcher. The material in Part I of this dissertation is work done jointly with Elette Boyle, Niv Gilboa, and Yuval Ishai. As collaborators, they were welcoming, encouraging, and forgiving, and they were so much fun to work with. Yuval's obsession with finding the right abstraction has changed the way I think about cryptography, and I continue to admire his consistent good humor and humility.

I have found that making mistakes is one of the best (if not the most pleasant) ways to learn. I really appreciate those who took the time to email me to report errors in my papers: Justin Holmgren, Elette Boyle, Jiamin Zhu, Ben Riva, and Ling Ren. Each of them brainstormed and checked candidate fixes with me, and each was unfailingly kind throughout the process. Justin Thaler offered a number of suggestions that improved the work in Part I of this dissertation. During the shepherding process at NSDI 2017, Jay Lorch gave extensive and extremely helpful editorial advice on the work that forms Chapter 4 of this dissertation.

I would like to thank Fraser Brown, Geoffrey Voelker, Lorenzo Alvisi, Mihir Bellare, Phil Rogaway, and Remzi Arpaci-Dusseau for offering sound and thoughtful counsel as I made decisions in the last year of graduate school.

The Stanford faculty has been extraordinarily supportive throughout my PhD studies. Matei Zaharia always had time for a technical discussion with me and offered good ideas faster than I could write them down. Phil Levis reminded me over and over to be true to my own taste and values in research. Zakir Durumeric shared his candid thoughts about starting out as a new professor. Keith Winstein was generous beyond all reason with his time, office space, funding, and controversial opinions. His critical eye improved my talks and made me a more careful researcher. Omer Reingold was the first person I would ask about difficult meta-research questions, and I always left his office happier than when I entered. David Mazières gave very candid and very helpful criticism of my talks and writing throughout grad school, and his encouragement during the job-application process was invaluable. David's mantras ("Close the loop!") will stay with me for life. A conversation with Martin Hellman early on in the PhD process shaped the way I viewed research, and I am delighted and thankful that he agreed to sit on the committee for my dissertation defense. I thank Nate Persily for agreeing to serve as the external chair for my defense.

When I was an undergrad, Bryan Ford first got me excited about systems and computer security. He has been my academic guardian angel ever since. Bryan also taught me the

most important lesson about research: if you're not having fun, you're doing it wrong. Long before Stanford, Maddie Hogan and Brie Regis demonstrated to me how much impact a good teacher can have on a student's life. Jim McNamara exposed me to the beauty (and pain) of open-source software. Rob Corrigan nurtured my early interest in computers, even after I accidentally destroyed one of his hard drives.

Dan Boneh's advising carried me through the PhD program. From the very beginning, Dan pushed me to pursue my own research interests, even when they diverged from his own. In the moments when I really needed help, Dan would jump in to provide a beautiful technical idea, some good counsel, or a flood of material support. Dan pushed me to be a better writer, a clearer thinker, and a more ambitious researcher. More important than all of that, the fact that Dan had confidence in me gave me confidence in myself. Thanks for all of this, Dan. You set the standard for advising that I hope some day to meet.

Finally, I want to thank my friends and family, including my extended family, for all of their love and support. My longtime friends Eric Ashkenas, Natasha Mevs-Korff, Zach Etzel, Alana Campbell, and Andrew Udelsman hosted me often at their homes during my grad-school travels and are just wonderful people to have in my life. Stefan Heule made time all the time to have a sushi dinner and to discuss any challenging technical or non-technical issue that was on my mind. Judith and Stanley Lubman were always up for a conversation about current events and they invited me to stay at their boathouse in Inverness countless times, which was a beautiful and peaceful place to think and write. My friends from Berkeley, from college, from grad school, and from my travels abroad made life outside of school entertaining and fun.

Joe Gibbs has been an unfailingly loyal cousin and friend and, in the great Gibbs tradition, makes sure that I always get enough to eat. Katherine and Ryan Stuppi's house in Menlo Park was a home away from home for me in grad school. My aunts, uncles, and cousins made me welcome whenever I was in the neighborhood and gave me lots of encouragement and support throughout the PhD process. Jane and David Acker, in particular, invited me to crash their vacations on many occasions. Hanging out with them in Europe taught me much about how to enjoy life.

It would be impossible for me to adequately thank my grandparents, Irwin and Marion Gibbs, John and Sylvia Corrigan, and Eileen Corrigan, for the role that they played in my upbringing and in my education. I feel so lucky to have had them in my life.

My sibling, Sara Corrigan-Gibbs, made sure that I was asking myself the tough questions at each career and personal junction, and Sara pushed me to make good choices at every step along the way.

My parents, Margaret Corrigan and Larry Gibbs, showed me by example that it is possible to do work that is intellectually challenging, that aims to further the goal of social justice, and that leaves time for fun and for family. Thank you, Mom and Dad, for all of your guidance and advice, and especially for your unconditional love.

The work in this dissertation was funded in part by an NSF Graduate Research Fellowship and an NDSEG Fellowship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We trust a small number of companies with vast amounts of our most private information. A single social network collects the personal messages and photos of more than two billion people [123]. A single medical-records system stores health data on more than half of U.S. medical patients [140]. And three Internet service providers transit communications for over 60% of U.S. broadband customers [27].

Centralization creates major economies of scale. At the same time, entrusting so few entities with so much data creates serious potential for abuse, both by dominant companies and by intruders who breach their defenses. Unfortunately, experience shows that where there is the potential for abuse, there *is* abuse—by attackers [169], by governments [74, 253], and by businesses [163, 259, 269].

In an ideal world, we could get the benefits of modern computing—global networks, web search, social networking, and so on—without needing to trust *anyone* to protect the privacy of our information. Upon reflection, though, it becomes quickly clear that unless we are going to build our own hardware, write our own compilers, and disconnect ourselves from the Internet, we must invest some amount of trust in systems that we did not construct and do not control [257].

It would seem, then, that we are trapped: computer systems are premised on trust—in the hardware, in the software, in the network—but placing blind trust in these components is precisely what puts the security of our data and our devices at risk.

In this dissertation, we demonstrate a way out. We show that it is possible to build large-scale systems that operate on sensitive data without needing to entrust that information to any single external party. In particular, we construct systems that protect privacy by

splitting trust among multiple parties in such a way that the failure of any one, whether benign or malicious, does not cause a catastrophic privacy failure for the system as a whole.

The idea of splitting trust is not new—it long predates this dissertation [33, 77, 78, 79, 146] and even predates the field of computer science [227]. Our contribution is to show that, for certain common tasks, it is possible to split trust in large-scale multi-user systems without sacrificing the performance and functionality properties that make these systems so useful in the first place.

By developing and applying a new cryptographic tool, *zero-knowledge proofs on distributed data*, we construct two systems that protect privacy by splitting trust. The first is a system that allows a service provider to collect aggregate statistical data about its user without learning any individual user's personal information. The second is a system for metadata-hiding communication that allows its users to communicate over an insecure network without revealing who is sending messages to whom.

Both of these systems maintain their security properties in the face of an attacker who can control the entire network, an unlimited number of participating users, and any proper subset of the servers that comprise the system. These systems split trust in the sense that, as long as an attacker cannot compromise *all* servers, the system provides "best-possible" protection of the confidentiality of the users' sensitive data.

We implement both systems and evaluate them on networks with nodes distributed around the world. In addition, our system for the private collection of aggregate statistics has been integrated into the Firefox web browser, and our code for the system has shipped to millions of Firefox users since December 2018.

Through the design, implementation, and deployment of these systems, we show that we can enjoy the benefits of modern computing while protecting the privacy of our data.

1.1 Overview of results

This dissertation is in two parts. In the first, we develop new cryptographic techniques for splitting trust. In the second, we apply these techniques to construct two new systems that protect privacy by splitting trust.

Part I: Cryptographic techniques for splitting trust

The first contribution of this dissertation is to define and construct *zero-knowledge proofs on distributed data*, a new type of interactive proof system. We will first introduce this primitive, then we will explain how it applies to the systems we build.

A traditional zero-knowledge proof [151] is an interaction between a computationally unbounded prover and a polynomial-time verifier. Both parties hold a common input string $x \in \{0,1\}^*$ and the prover's challenge is to convince the verifier that the input x is in some language $\mathcal{L} \subseteq \{0,1\}^*$, without leaking anything else about x to the verifier. For example, the prover might try to convince the verifier that x is the binary representation of an integer that has exactly two prime factors, without revealing these prime factors to the verifier. An astonishing fact is that, under the minimal assumption that one-way functions exist, every language in NP has a zero-knowledge proof system [147].

We are interested in a twist on the typical setting of zero-knowledge proofs. In our setting, there are *multiple* verifiers, and each verifier holds only a *piece* of the input string x. Even though neither verifier holds the input x in its entirety, the prover still must convince the verifiers that this string, which they hold in *distributed fashion*, is in some language $\mathcal{L} \subseteq \{0,1\}^*$.

As in a standard zero-knowledge proof, the verifiers should learn nothing about the string x apart from the fact that $x \in \mathcal{L}$. Furthermore, neither verifier should learn anything about the other verifier's piece of the input by participating in this protocol, apart from what they can infer from the fact that $x \in \mathcal{L}$. (To formalize this notion of privacy, we extend the standard simulation-based notion of zero knowledge.) We call this new type of proof system a *zero-knowledge proof on distributed data*.

Let us now explain why zero-knowledge proofs on distributed data are instrumental to the privacy-preserving systems we construct in this dissertation. In each of these systems, a small set of infrastructure servers collects and processes data from a large set of potentially adversarial clients. Speaking very informally, we want these systems to maintain two properties at once:

1. ***Privacy* against malicious servers.** If *any one* server behaves honestly, then even a coalition of adversarial clients and servers should not be able to compromise the privacy of honest clients' data.
2. ***Robustness* against malicious clients.** If *all* servers behave honestly, then even a

coalition of adversarial clients should not be able to disrupt the functioning of the system.

In both systems, we achieve the *privacy* property using cryptographic secret sharing [48, 247]: the client splits its data using a secret-sharing scheme and sends one share to each server. The properties of the secret-sharing scheme imply that an adversary must compromise *all* servers to recover any user's private input. Furthermore, we show that it is possible for the servers to collectively perform non-trivial computations on the clients' data, even though the clients' data remains in secret-shared form.

However, our use of secret-sharing makes the *robustness* property difficult to provide: if each server only sees a single share of the client's data, no server can unilaterally determine whether a particular client has sent to the servers a "well-formed" data submission, or whether the client has sent a maliciously crafted submission that would corrupt the output of the computation.

This is exactly where our new zero-knowledge proofs on distributed data apply. Using these proofs, a client (holding an input $x \in \{0,1\}^*$) can convince a set of servers (each holding a share of x) that x is the language $\mathcal{L} \subseteq \{0,1\}^*$ of "well-formed" submissions. Furthermore, the client can prove this to the servers without leaking *any* additional information to the servers about its private submission x.

The notion of "well-formedness" differs in the two different systems, as does the nature of the computation that the servers perform on the clients' secret-shared data. However, in both systems we apply our new zero-knowledge proofs to preserve privacy without compromising robustness.

It is possible to construct zero-knowledge proofs on distributed data using existing cryptographic techniques, such as dishonest-majority multi-party computation protocols [146] or standard zero-knowledge proofs [151]. However, these techniques require cryptographic assumptions and, as we will show, they are relatively inefficient in concrete terms. Our proof systems are information-theoretically secure (i.e., they require no computational assumptions) and they are concretely quite efficient for both the prover and verifiers.

The primary limitation of our basic proof systems is that their communication complexity is relatively large. For an arbitrary language $\mathcal{L} \subseteq \{0,1\}^*$, the total communication complexity of our proof system on distributed data for $\mathcal{L}$ grows *linearly* with the size of a circuit for computing $\mathcal{L}$. In contrast, succinct zero-knowledge techniques can achieve communication

complexity *poly-logarithmic* or even *constant* in the circuit size, at the cost of making computational assumptions [46, 47, 55, 60, 134, 156, 170, 201, 210, 226, 238, 240, 266, 281]. To address this limitation of our basic proof systems, we show that for many interesting special cases—including those that arise in our privacy-preserving systems—it is also possible to reduce the proof size to poly-logarithmic or even constant, while still using only fast information-theoretic techniques.

We construct zero-knowledge proofs on distributed data in two steps. First, in Chapter 2, we define *fully linear probabilistically checkable proofs* ("fully linear PCPs"), a new type of information-theoretic proof system. Next, in Chapter 3, we show that if a language $\mathcal{L}$ has a fully linear PCP with short proofs, it also has a communication-efficient zero-knowledge proof system on distributed data. Thus, by constructing fully linear PCPs with short proofs, we can construct communication-efficient zero-knowledge proofs on distributed data. We also consider an interactive analogue of fully linear PCPs and show that these too imply zero-knowledge proofs on distributed data.

To instantiate this framework, we show that a number of widely used information-theoretic proof systems in the cryptographic literature [47, 134, 149, 170, 268] give implicit constructions of fully linear proof systems. In addition, we also construct new fully linear proof systems for simple or structured languages—including the languages that arise in our applications—that have short proofs. Plugging our new fully linear proof systems into this general framework yields a family of communication-efficient zero-knowledge proofs on distributed data.

Part II: Systems that split trust to protect privacy

In the second part of this dissertation, we describe the design and implementation of two systems that protect the privacy of user data by splitting trust. Both of these systems exploit our zero-knowledge proofs on distributed data to yield multiple orders-of-magnitude speed-ups over the prior work.

Prio: Private computation of aggregate statistics. Our digital devices constantly gather data about their surroundings and send this data back to the manufacturer for analysis: mobile apps report location information, connected thermostats report energy usage, and modern cars report speed and driving behavior. Often device manufacturers collect this information for the purpose of computing *aggregate* statistics about their user population

as a whole. For example, the provider of a real-time traffic app wants to know "How much traffic is there on the Bay Bridge?" but the provider is not interested in learning the location of any individual app user.

Today, manufacturers usually compute these statistics by collecting disaggregated data from their users directly, and storing it for analysis and aggregation later on. While this approach is easy to implement, it puts the users' sensitive data at risk of theft by attackers, abuse by companies, and seizure by governments.

The Prio system, which is the topic of Chapter 4, allows a device manufacturer to compute aggregate statistics over sensitive user data *without* ever seeing the disaggregated user data. Prior systems for private aggregation either produce noisy approximations of the aggregate statistic [121, 124], do not protect against data-corruption attacks by malicious clients [119, 172, 188, 207, 208], or rely on relatively costly public-key cryptographic machinery [119, 231].

Prio naturally applies to scenarios in which it is sufficient to learn aggregate statistics—rather than targeted user-level data—about a population. For example, Prio is well-suited to telemetry applications in which the manufacturer of a device or app wants to learn how its customers are using its product. Prio is *not* suited to applications in which it is necessary to obtain disaggregated personalized statistics, as might be needed to implement, for instance, a system for privacy-preserving ad-targeting or personalized medicine.

A Prio deployment consists of a small number of servers and a large number of clients. Each client $i \in \{1, \ldots, n\}$ holds a data point x_i in some domain $\mathcal{D}$. The configuration of the system specifies a public aggregation function $f \colon \mathcal{D}^n \to \mathcal{R}$ and the servers' goal is to learn the value of the statistic $f(x_1, \ldots, x_n)$, computed over all clients' data. For example, the value x_i could indicate the number of hours that client i spent using a particular app in a month and the function $f(x_1, \ldots, x_n)$ could output the average of these numbers—the monthly usage of the app averaged across all users. Prio supports a wide range of aggregation functions f, including SUM, AVERAGE, STDDEV, MOST-POPULAR (approximate), and LINEAR-REGRESSION.

As long as at least one Prio server is honest, the only information that the servers learn during a protocol run is the value of the aggregate statistic $f(x_1, \ldots, x_n)$ that the system computes. (For some aggregation functions, the system also reveals a handful of auxiliary values, but this leakage is limited and quantifiable.) The system is *robust* against malicious clients as well: the worst that a malicious client can do to corrupt the system's output is to lie about the value of its private input value x_i within predetermined bounds that the servers set.

The system is performant, even with a large number of clients: a small deployment of five geographically dispersed Prio servers can handle hundreds of client submissions per second.

To briefly sketch how Prio works, consider a deployment of the system with only two servers in which we aim to collect the sum of n client-provided integers $x_1, \ldots, x_n \in \{0, 1\}$. Even this simple statistic is already useful: an app developer could use it to privately learn, for example, how many clients used a particular feature of the app. We aim to protect the privacy of the clients' data as long as at least one of the two Prio servers is honest.

To submit data to the system, each client i chooses a random value $r_i \xleftarrow{R} \mathbb{Z}_p$, where p is a fixed prime greater than the number of clients n. The client then sends:

- $r_i \in \mathbb{Z}_p$ to the left server and
- $x_i - r_i \in \mathbb{Z}_p$ to the right server.

These values consist of additive secret shares of the client's private data value x_i.

After collecting submissions from all n clients, the left server publishes the sum of the values it has received: $A_{\text{left}} \leftarrow \sum_{i=1}^{n} r_i \in \mathbb{Z}_p$. The right server similarly publishes the sum of the values it has received: $A_{\text{right}} \leftarrow \sum_{i=1}^{n} (x_i - r_i) \in \mathbb{Z}_p$. By summing these two published values modulo p, the servers recover the sum of the x_is—the aggregate statistic of interest:

$$A_{\text{left}} + A_{\text{right}} = \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} (x_i - r_i) = \sum_{i=1}^{n} x_i \qquad \in \mathbb{Z}_p.$$

As long as one of the two servers is honest, the remaining server learns nothing about any client's private value x_i, apart from what it can infer from the aggregate statistic $\sum_{i=1}^{n} x_i$. Furthermore, if the servers add a small amount of noise to their values A_{left} and A_{right}, the system can achieve differential privacy [115, 117] as well.

There is one major problem with the simple scheme we have sketched: if one client deviates from the protocol by sending two independent random values to the two servers, the client can completely corrupt the output of the protocol. In addition, the servers will not know which client mounted this attack—the privacy of the system protects the privacy of the attacker.

We solve this problem in Prio using the machinery we have developed for zero-knowledge proofs on distributed data. To do so, we have the client send not only the encoding of its data value x_i to each server, but we also have the client prove to the servers, in zero knowledge, that it has sent them values v_{left} and v_{right} such that $v_{\text{left}} + v_{\text{right}} \in \{0, 1\} \in \mathbb{Z}_p$.

The soundness property of the proof system ensures that no matter how the client attempts to cheat, the servers will reject a malformed submission with overwhelming probability. The zero-knowledge property of the proof system implies that even if one of the two servers is malicious and deviates from the protocol, it learns nothing about the client's private value x_i. Finally, since our proofs on distributed data are information-theoretic and lightweight, the computational overhead of adding these proofs to the system is minimal.

The system we have sketched allows the servers to privately compute sums of integers, but we would like to compute more sophisticated statistics as well. To capture more complex statistics we have the client encode its data in a more involved manner, using ideas from streaming algorithms [75, 91, 160]. The servers then use the simple scheme we have sketched to compute sums of these encodings. We then show that it is possible to recover the statistic of interest from the sum of the encodings. (See Section 4.5 for details.)

By combining all of these ideas in Prio, we are able to collect interesting aggregate statistics with strong privacy guarantees, while protecting against against malicious clients, and at modest computational cost.

Riposte: Anonymous messaging at million-user scale. Network surveillance has become ubiquitous [43, 132, 133, 142, 215]. Today, sending a message from one point on the Internet to another creates a record of that communication, which may be indelible. And yet, we would like to protect socially valuable anonymous communication. We would like, for example, to allow whistleblowers to anonymously report abuses of power to members of Congress without fear that their communications metadata could later compromise their privacy.

Riposte, the topic of Chapter 5, is a system for anonymous communication that allows participants to anonymously post messages to a public bulletin board, maintained by a small set of servers. The system provides strong anonymity guarantees in that, as long as at least one of the system's servers executes the protocol correctly, even an adversary who controls the network, a subset of the servers, and a large coalition of users cannot determine which honest user posted which message.

Prior anonymity systems either lacked protection against this form of strong network adversary [106, 112, 129, 166, 197, 235] or scaled to relatively small anonymity-set sizes [141, 274]. For example, the widely used and tremendously successful Tor system for low-latency anonymous browsing [112] is vulnerable to a variety of traffic-analysis attacks [22, 213, 214]. In contrast, for low-bandwidth, latency-tolerant applications—such as messaging or

whistleblowing—Riposte provides provable protection against traffic-analysis attacks while achieving anonymity-set sizes of millions of users (Section 5.6).

To explain how Riposte works, we will sketch a simplified two-server version of the system. The Riposte servers collectively maintain a bulletin board such that the clients can anonymously post messages to this bulletin board. In effect, the servers maintain a database into which the clients can anonymously write. Or equivalently, think of the servers as maintaining a database such that the clients can anonymously write into this database. We think of each database row as holding an element of a finite field $\mathbb{F}$, so we can represent the state of an L-row database as a vector in in $\mathbb{F}^L$.

To write a message $m \in \mathbb{F}$ into the database, a client first samples the index of a random database row $\ell \xleftarrow{R} \{1, \ldots, L\}$ and a random vector $r \xleftarrow{R} \mathbb{F}^L$. Then, the client takes the all-zeros vector in $\mathbb{F}^L$ and adds its message m into the ℓ-th coordinate. Denote the resulting vector as $m \cdot e_\ell \in \mathbb{F}^L$. Finally, the client sends the vector $r \in \mathbb{F}^L$ to the left server and $m \cdot e_\ell - r \in \mathbb{F}^L$ to the right server.

After collecting many of these write requests from many clients, the servers can reveal the plaintext state of the database. To do so, each server publishes the sum of the messages it has received from the clients. If there are n clients total, the left server publishes $D_{\text{left}} = \sum_{i=1}^{n} r_i \in \mathbb{F}^L$ and the right server publishes $D_{\text{right}} = \sum_{i=1}^{n} (r_i - m_i \cdot e_{\ell_i}) \in \mathbb{F}^L$. (Here we use $r_i \in \mathbb{F}^L$ and $m_i \in \mathbb{F}$ to denote client i's randomness and message, and $\ell_i \in \{1, \ldots, L\}$ to denote the database location into which client i writes.)

By combining their two values, the servers recover the plaintext database state:

$$D_{\text{left}} + D_{\text{right}} = \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} (r_i - m_i \cdot e_{\ell_i}) = \sum_{i=1}^{n} m_i \cdot e_{\ell_i} \quad \in \mathbb{F}^L \,,$$

which has the ith client's message m_i in location ℓ_i of the database.

Even this very simple scheme is already non-trivial: multiple clients can anonymously write into the database collectively held at the servers. As long as at least one server is honest, an adversary controlling the rest of the servers, any number of clients, and the entire network, cannot tell which honest client wrote which message into the database.

Unfortunately, the simple scheme has a number of drawbacks.

First, *the communication cost is large.* Each client must upload a string to each server that is as large as the entire database. To address this problem, we apply an idea from

the literature on private information retrieval [85]. A private information retrieval system allows a client to *read* a row from a database (in a communication-efficient way) without revealing which row it is reading. By running a private information retrieval protocol in reverse, we show in Section 5.4 that it is possible for a client to *write* into a database (in a communication-efficient way) without revealing which row it is writing. This idea, along with recent developments in protocols for private information retrieval [62, 63, 138], can drop the communication cost of this scheme we sketched from the original $2L$ bits down to $O(\sqrt{L})$ or even to the optimal $O(\log L)$ bits. (Here, we take the field size to be a constant.)

The second drawback of the simple scheme is that *a single client can anonymously corrupt the database state.* As in Prio, the proper functioning of the system relies on the fact that every client sends correlated messages to the two servers. A client who sends independent random messages to the two servers could completely corrupt the output of the system. As in Prio, we can address this problem in Riposte using our zero-knowledge proofs on distributed data. After a client submits its write request to the two servers, the client proves, in zero knowledge, to the servers that it has sent them additive shares of a vector in $\mathbb{F}^L$ that is zero everywhere except at one coordinate. The soundness property of the proof system ensures that the Riposte servers can always catch disruptive clients. The zero-knowledge property ensures that a malicious coalition of servers learns nothing about the client's private message. Moreover, the proof consists of only a *constant* number of field elements, independent of the length L of the database, so that the total communication cost between the client and servers remains quite small.

The final drawback of the simple scheme is that *multiple honest clients could write into the same database row.* These "colliding writes" would render both clients' messages unrecoverable. To ameliorate this problem, we show how clients can encode their messages in such a way that allows message recovery even if multiple writes collide in the table.

Combining these design elements yields a system for anonymous messaging that scales to support millions of users—orders of magnitude more than prior systems could handle—for latency-tolerant applications.

1.2 Impact in practice

Mozilla has built Prio into the Firefox web browser and is planning to use the system to collect sensitive telemetry data from their users in a privacy-preserving way [120, 162]. As of

December 2018, the Prio code ships to *every* Firefox user—hundreds of millions of machines. This constitutes the largest deployment of technology based on probabilistically checkable proofs to date.

By way of background, the Firefox browser has a feature called "Content Blocking" [70] that aims to prevent websites from loading known tools for cross-site tracking, such as fingerprinting scripts. Content Blocking is a useful privacy feature, since it prevents advertisers and data brokers from following the activities of a person as she browses from one website to the next. However, Content Blocking can inadvertently block benign JavaScript from loading and can prevent a website from rendering properly. When this happens, the user can click a button in the browser's user interface to disable Content Blocking.

To debug the content-blocking feature, Mozilla engineers would like to learn the list of websites on which users most often disable Content Blocking. This list would reveal which popular websites Content Blocking breaks, which would be useful debugging information for the engineers. However, Mozilla would like to collect this aggregate information without learning anything about which user visited which website.

This is an instance of the private-aggregation problem. For each website in the top 1000 websites, each Firefox user i has a bit $x_i \in \{0, 1\}$ indicating whether it has disabled Content Blocking for that site in the past 24 hours. These bits x_i are sensitive, since they encode information about the user's browsing history. For each site, Mozilla would like to know how many users have disabled Content Blocking on that site, which necessitates computing a sum $\sum_{i=1}^{n} x_i$ over all n users of the Firefox browser. Mozilla's goal is to compute this sum without learning any individual user's private value x_i.

Mozilla is using Prio to solve this private-aggregation problem. As of version 64 of Firefox (released in December 2018), the browser ships with libprio, a C library we wrote that implements a simplified Prio client. (The code is available under an open-source license at https://github.com/mozilla/libprio/.) Periodically, the browser uses libprio to generate two encrypted packets—one for each of two Prio servers—and uploads these packets through Mozilla's existing telemetry system. From there, a pair of Prio servers can pull these encrypted packets out of Mozilla's telemetry infrastructure, process the packets, and compute the desired aggregate statistics.

As of October 2019, Mozilla is running both of the Prio servers so the system does not yet split trust across organizations. For that reason, Mozilla is currently only using the system to collect non-sensitive user information, and the Prio client is only enabled by default on

the Nightly and Beta release channels of the browser. Still, there are some millions of users on these two release channels. Mozilla is in discussions now with potential external partners towards the goal of having a separate entity run the second Prio server. Once this last step is complete, Mozilla will be able to collect this aggregate content-blocking data from its users without ever having access to any disaggregated user data.

Once the deployment of Prio in Firefox is complete, we hope it will demonstrate to other companies that by splitting trust it *is* possible to process sensitive user information at large scale in a privacy-preserving way.

Bibliographic notes

This dissertation is based on the following jointly authored publications:

Chapters 2 and 3: "Zero-knowledge proofs on secret-shared data via fully linear PCPs," with Dan Boneh, Elette Boyle, Niv Gilboa, and Yuval Ishai, published at the IACR Annual International Cryptology Conference ("CRYPTO") in 2019 [54],

Chapter 4: "Prio: Private, robust, and scalable computation of aggregate statistics," with Dan Boneh, which appeared at the USENIX Symposium on Networked Systems Design and Implementation in 2017 [93], and

Chapter 5: "Riposte: An anonymous messaging system handling millions of users," with Dan Boneh David Mazières, which appeared at the IEEE Symposium on Security and Privacy in 2015 [94].

Any errors or omissions in this dissertation are, of course, mine alone.

Notation

We will use the following notation throughout this dissertation.

Sets. For a finite set S, the notation $x \xleftarrow{R} S$ indicates that the value of x is sampled independently and uniformly at random from S. The notation $\{x_i\}_{i \in S}$ indicates the unordered set $\{x_i \mid i \in S\}$. For a positive integer n, we use the shorthand $[n] = \{1, \ldots, n\}$ and we let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the set of natural numbers.

Algebra. We let $\mathbb{Z}_m$ denote the ring of integers modulo m—i.e., the integers with addition and multiplication modulo m. We use $\mathbb{F}$ to denote a finite field. For concreteness, think of $\mathbb{F}$ as the set of integers with addition and multiplication modulo a fixed prime. There are other fields that are useful in cryptographic applications, but these are more complicated to describe and are not important for the applications in this dissertation.

Vectors. If $\mathbb{F}$ is a finite field and $x \in \mathbb{F}^n$ and $y \in \mathbb{F}^m$ are vectors of field elements, we denote their concatenation as $(x \| y) \in \mathbb{F}^{n+m}$ and if $n = m$, we denote their inner product as $\langle x, y \rangle$, so $\langle x, y \rangle = \big\langle (x_1, \ldots, x_n), (y_1, \ldots, y_n) \big\rangle = \sum_{i=1}^{n} x_i y_i \in \mathbb{F}$. We use e_ℓ to represent a vector that is zero everywhere except at index ℓ, where it has value "1." The underlying vector space should be clear from the context. Thus, for $m \in \mathbb{F}$, the vector $m \cdot e_\ell \in \mathbb{F}^L$ is the vector whose value is zero everywhere except at index ℓ, where it has value m. In the chapter on Prio (Chapter 4), we use the notation $[\![x]\!]_j$ to denote an additive share of a value $x \in \mathbb{F}$, or a vector $x \in \mathbb{F}^k$.

Algorithms and indistinguishability. We use the notation "$\stackrel{\text{def}}{=}$" to define a new function or symbol and we use $\perp$ to denote the empty string. We use $x \leftarrow 7$ to indicate assignment. We use $\text{polylog}(n)$ to indicate a fixed polynomial in $\log n$ and all logarithms are base two, unless otherwise specified. When $\mathcal{D}_0$ and $\mathcal{D}_1$ are probability distributions with finite support, we write $\mathcal{D}_0 \equiv \mathcal{D}_1$ to indicate that they are equivalent.

Arithmetic circuits. An arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$ takes as input a vector $x = \langle x_1, \ldots, x_n \rangle \in \mathbb{F}^n$ and produces a single field element as output. We represent the circuit as a directed acyclic graph, in which each vertex in the graph is either an *input*, a *gate*, or an *output* vertex. Input vertices have in-degree zero and are labeled with a variable in $\{x_1, \ldots, x_n\}$ or a constant in $\mathbb{F}$. Gate vertices have in-degree two and are labeled with the

operation + or ×. The circuit has a single output vertex, which has out-degree zero. To compute the circuit $\mathcal{C}(x) = \mathcal{C}(x_1, \ldots, x_n)$, we walk through the circuit from inputs to outputs, assigning a value in $\mathbb{F}$ to each wire until we have a value on the output wire, which is the value of $\mathcal{C}(x)$. In this way, the circuit implements a mapping $\mathcal{C} : \mathbb{F}^n \to \mathbb{F}$. When C is an arithmetic circuit over a finite field, we use $|C|$ to denote the number of multiplication gates in the circuit, not counting multiplications by constants, unless otherwise specified. We can represent an arithmetic circuit $C(x_1, \ldots, x_n)$ over $\mathbb{F}$ as an n-variate polynomial over $\mathbb{F}$; we let $\deg C$ denote the total degree of this polynomial.

Part I

Techniques

In this part of the dissertation, we develop new techniques for proving in zero knowledge statements that are *distributed* (i.e., partitioned or secret-shared) across two or more verifiers. Recall that in a standard interactive proof system [16, 18, 32, 151] a verifier holds an input $x \in \{0,1\}^*$ and a prover tries to convince the verifier that x is a member of some language $\mathcal{L} \subseteq \{0,1\}^*$. We consider instead the setting in which there are *multiple* verifiers, and each verifier holds only a piece of the input, such as a share of x generated using a linear secret-sharing scheme. Critically, no single verifier holds the entire input x. The prover, who holds the entire input x, must convince the verifiers, who only hold pieces of x, that $x \in \mathcal{L}$. At the same time, we require that the proof system be *strongly zero knowledge*: every proper subset of the verifiers should learn nothing about x, apart from the fact that $x \in \mathcal{L}$.

This type of proof system directly applies to the systems that we construct for private computation of aggregate statistics and for anonymous messaging in the second part of this dissertation. These proof systems also find application beyond this dissertation, to private ad targeting [258], to verifiable function secret sharing [63], and to malicious-secure multi-party computation [54].

In Chapter 2, we introduce the central new abstraction of a *fully linear proof system*, which we use in Chapter 3 to construct zero-knowledge proof systems on distributed and secret-shared data.

We now give an overview of the contributions in this first part of the dissertation.

Fully linear proof systems. We begin in Chapter 2 by introducing the notion of a *fully linear proof system*, which captures the information-theoretic object at the core of all of our constructions. We consider the non-interactive variant of such proof systems, called *fully linear PCPs*, and then we describe a natural extension to the interactive setting.

A fully linear PCP is a refinement of standard linear PCPs [14, 47, 170]. In a standard linear PCP over a finite field $\mathbb{F}$, a polynomial-time verifier holds an input $x \in \mathbb{F}^n$ and a prover produces a proof $\pi \in \mathbb{F}^m$ to the assertion that $x \in \mathcal{L}$, for some language $\mathcal{L} \subseteq \mathbb{F}^n$. The verifier checks the proof by reading x and making *linear queries* (i.e., inner-product queries) to the proof π. In particular, the verifier can make a bounded number of queries to the proof of the form $q_j \in \mathbb{F}^m$, and receives answers $a_j = \langle q_j, \pi \rangle \in \mathbb{F}$.

In a fully linear PCP, we further restrict the verifier: the verifier cannot read the entire input x directly, but only has access to it via linear queries. Concretely, the verifier in a fully linear PCP makes linear queries q_j to the concatenated input-proof vector $(x \| \pi) \in \mathbb{F}^{n+m}$

and must accept or reject the assertion that $x \in \mathcal{L}$ based on the answers a_j to these linear queries. Motivated by the applications we consider, we would also like fully linear PCPs to satisfy the following *strong zero-knowledge* requirement: the queries q_j together with the answers a_j reveal no additional information about x other than the fact that $x \in \mathcal{L}$. This is stronger than the standard notion of zero knowledge, in which the verifier learns the input x in its entirety. In contrast, in our setting, the verifier not learn x itself—the verifier only learns that it has *linear oracle access* to an $x \in \mathcal{L}$.

In many cryptographic applications, no single verifier holds the entire input statement x, but the verifiers can jointly access it via linear queries. This situation arises in scenarios in which the input x is distributed or secret-shared between two or more parties, or when the input is encoded using an additively homomorphic encryption or commitment scheme. In these scenarios, verifiers can readily compute answers to public linear queries via *local* computations on their views of x. These settings naturally motivate the full linearity restriction on the proof systems we construct. While fully linear PCPs can be meaningfully applied in all of the above scenarios, we will focus on their applications to proofs on distributed or secret-shared data.

We stress again that in a fully linear PCP, the verifier only has *linear query access* to the input x. An interesting consequence is that even if $\mathcal{L}$ is an easy language decidable in polynomial time, a verifier making a bounded (e.g., constant) number of such queries typically cannot decide whether $x \in \mathcal{L}$ without the aid of a proof, even if the verifier can run in unbounded time. This makes the existence of fully linear proof systems with good parameters meaningful even for finite languages and even if, say, P = PSPACE.

The fact that even easy languages can be hard to decide in our setting is similar to the situation that arises with *proofs of proximity* [42], which place a more stringent restriction on the verifier's access to the input. However, unlike proofs of proximity, in fully linear PCPs the verifier is guaranteed that the input is actually in the language rather than being "close" to some input the language. Another related notion is that of a *holographic proof* [17, 161], in which the verifier gets oracle access to an *encoding* of the input using an arbitrary error-correcting code.

The verifier's restricted access to the input x also makes possible a connection between fully linear PCPs and communication complexity [9, 185, 189]. Using this connection, we prove unconditional lower bounds on the efficiency properties of fully linear PCPs (Section 2.5).

The vast literature on succinct arguments for NP languages makes extensive use of linear

PCPs, both explicitly and implicitly [46, 47, 55, 60, 134, 156, 170, 201, 226, 238, 240, 266, 281]. We can cast these existing linear PCPs, including the so-called Hadamard PCP [14, 170] and ones obtained from quadratic span programs or quadratic arithmetic programs [47, 134, 225], into the fully linear framework. Prior systems for proof on committed or secret-shared data has implicitly used this fact [19, 93, 97]. Our notion of fully linear PCPs makes explicit the properties that a linear PCP must satisfy to be useful in these applications.

Shorter proofs for structured and simple languages. When using fully linear PCPs to build zero-knowledge proof systems on distributed or secret-shared data the *proof length* determines the number of bits that the prover must send to the verifiers. As such, we aim to design fully linear PCPs with short proofs.

For general NP relations, all known linear PCPs have size at least *linear* in the size of an arithmetic circuit recognizing the relation. In Section 2.3, we achieve significant length savings by designing new *sublinear* sized fully linear PCPs for languages recognized by deterministic circuits with repeated sub-structures (Theorem 2.3.3) or by a degree-two polynomial (Corollary 2.3.7). In the latter case, we can even prove that the $O(\sqrt{n})$ complexity of our construction is optimal up to low-order terms (Section 2.5). These and other proof systems constructed in this work satisfy the notion of strong zero knowledge that we introduced above.

Theorem 1.2.1 (Informal - Short fully linear PCPs for degree-two polynomials)**.** *If there is a single degree-two polynomial that recognizes membership in $\mathcal{L} \subseteq \mathbb{F}^n$, then $\mathcal{L}$ admits a fully linear PCP with strong zero knowledge that has proof length, query complexity $\tilde{O}(\sqrt{n})$, soundness error $O(\sqrt{n}/|\mathbb{F}|)$. Furthermore, there exists a language $\mathcal{L}$ as above such that the sum of the proof length and query complexity must be $\Omega(\sqrt{n})$, for a constant field size, even when we allow constant soundness error and do not require zero knowledge.*

See Corollary 2.3.7 and Theorem 2.5.1 for more precise and general statements.

Table 1.1 summarizes the communication and round complexity of the proof systems on secret-shared data for languages that frequently come up in practice, for example in the Prio system (Chapter 4) for privately aggregating data, and in the Riposte (Chapter 5) system for anonymous communication. The table illustrates the strong benefits of interactive fully linear proof systems over non-interactive ones.

Reducing proof size by interaction. To further drive down the proof length, we consider a generalization of fully linear PCPs that allows multiple rounds of interaction between

Language	Proof system	Comm. complexity	Rounds
Hamming weight 1:	Theorem 2.3.3	$O(n)$	1
$\bar{x} \in \mathbb{F}^n, \text{weight}(\bar{x}) = 1$	Corollary 2.4.5	$O(\sqrt{n})$	2
	Corollary 2.4.6	$O(\log n)$	$O(\log n)$
	Theorem 2.4.12	$O(1)$	2
$\bar{x} \in \{0, \ldots, B\}^n \subseteq \mathbb{F}^n$	Theorem 2.3.3	$O(B \cdot n)$	1
	Corollary 2.4.5	$O(B \cdot \sqrt{n})$	2
	Corollary 2.4.6	$O(B \cdot \log n)$	$O(\log n)$
Degree-two circuit	Theorem 2.3.3	$O(n)$	1
	Theorem 2.4.9	$O(\sqrt{n})$	2
	Theorem 2.4.9	$O(\log n)$	$O(\log n)$
Arbitrary circuit C, $C(\bar{x}) = 1$	Theorem 2.3.3	$O(n)$	1
(size n, depth d, fan-in 2)	Theorem 2.4.10 via GKR [149]	$O(d \log n)$	$O(d \log n)$

Table 1.1: Complexity of fully linear proof systems. We assume the proofs are over a finite field $\mathbb{F}$ with $|\mathbb{F}| \gg n$.
All systems in the table, except GKR, provide strong honest-verifier zero knowledge.

the prover and verifier. These *fully linear interactive oracle proofs*, or fully linear IOPs, are the linear analogue of interactive oracle proofs (IOP) [39], also known as probabilistically checkable interactive proofs [234]. We note that without the zero-knowledge requirement, it is possible to cast several existing interactive proof systems from the literature, including the GKR protocol [148], the CMT protocol [90], and the RRR protocol [234] into the form of fully linear IOPs.

For the case of "well-structured" languages, we show in Section 2.4 that interaction can dramatically shrink the proof size, while maintaining the required strong zero-knowledge property. In particular, any language whose membership can be verified by a system of constant-degree equations over a finite field admits a fully linear IOP with strong zero-knowledge in $O(\log n)$ rounds and only $O(\log n)$ proof length, provided that the underlying field is sufficiently large. Even for degree-two languages, this gives an exponential reduction in proof size over the non-interactive case.

Theorem 1.2.2 (Informal - Fully linear zero-knowledge IOPs for low-degree languages)**.** *If a system of constant-degree equations decides a language $\mathcal{L} \subseteq \mathbb{F}^n$ then $\mathcal{L}$ admits a fully linear IOP with strong zero knowledge, $O(\log n)$ rounds, proof length $O(\log n)$, and query complexity $O(\log n)$.*

See Theorem 2.4.9 for a more precise and general statement.

Zero-knowledge proofs on distributed or secret-shared data. In a proof on distributed (or secret-shared) data, the prover holds an input x and each of s verifiers $V_1, \ldots, V_s$ holds only a piece (or a secret share) of x. The prover's goal is to convince the verifiers that x is in some language $\mathcal{L}$, even though no verifier holds x in its entirety.

In Chapter 3, we show that it is possible to compile any fully linear PCP or IOP into a zero-knowledge proof system on distributed or secret-shared data in the following natural way. Instead of sending a proof vector π to a single verifier, the prover secret-shares the proof vector π between the s verifiers using a linear secret-sharing scheme. The verifiers can now locally apply each linear query to the concatenation of their share of the input x and their share of π. Each verifier then exchanges the resulting answer shares with the other verifiers. The verifiers then reconstruct the answers to the linear queries and apply the decision predicate to decide to accept or reject x. We show how to achieve zero knowledge in this setting when even all but one of the verifiers is malicious.

Theorem 1.2.3 (Informal - Distributed zero-knowledge proofs for low-degree languages on secret-shared data)**.** *If a system of constant-degree equations decides a language $\mathcal{L} \subseteq \mathbb{F}^n$ then, assuming ideal coin-tossing, there is an $O(\log n)$-round distributed zero-knowledge protocol for proving that $x \in \mathcal{L}$, where x is additively shared between s verifiers, with communication complexity $O(s \log n)$. The protocol is sound against a malicious prover and is strongly zero-knowledge against $s-1$ malicious verifiers.*

See Corollary 3.2.2 for a more precise and general statement. We also give a Fiat-Shamir-style compiler [126] that uses a random oracle to collapse multiple rounds of interaction into a single message sent by P to each V_j over a private channel, followed by a single message by each V_j.

We note that recent work [186, 216] also studied interactive proofs with distributed verifiers for the purpose of proving properties of a communication graph connecting a large number of verifiers. These works also observe the relevance of the interactive proofs of the GKR [148] and RRR [234] protocols to this distributed-verifier setting. Our focus here is quite different; we are motivated by the goal of proving in zero knowledge simple properties of data distributed among a small set of verifiers. As a result, our abstractions, constructions, and applications are very different from those in prior work [186, 216].

Chapter 2

Fully linear proof systems

2.1 A taxonomy of information-theoretic proof systems

One of the contributions of this work is to introduce and formalize the notions of *fully linear* PCPs and IOPs. To situate these new types of proof systems in the context of prior work, we briefly survey the landscape of existing proof systems. This discussion will be relatively informal; see Section 2.2 for formal definitions of linear and fully linear proof systems.

A tremendously successful paradigm for the construction of cryptographic proof systems is the following: First, construct a proof system that provides the security guarantees (e.g., soundness and zero-knowledge) against *computationally unbounded* adversaries. We will refer to this as an "information-theoretic proof system," or a "probabilistically checkable proof" (PCP). This information-theoretic system is often useless as a standalone object, since it typically makes idealized assumptions that are difficult to enforce in practice. For example, we might assume that certain pairs of messages are independent or that the verifier has restricted access to the proof.

Next, use cryptographic assumptions or an augmented model of computation (e.g., the random-oracle model [29]) to "compile" the information-theoretic proof system into one with a concrete implementation. This compiler might also eliminate interaction, improve communication complexity, or even provide an extra zero-knowledge property. The cost of this compilation step is that the resulting proof system may have security only against a *computationally bounded* prover and/or verifier. We refer to this type of compiler as a "cryptographic compiler."

Different kinds of information-theoretic proof systems call for different cryptographic

compilers. The main advantage of this separation is modularity: it is possible to design, analyze, and optimize information-theoretic proof systems independently of the cryptographic compilers. It may be beneficial to apply different cryptographic compilers to the same information-theoretic proof system, as different compilers may have incomparable efficiency and security features. For instance, they may trade succinctness for better computational complexity or post-quantum security.

To give just a few examples of this methodology: Micali [209] uses a random oracle to compile any classical PCP into a *succinct* non-interactive argument system for NP. As another example, Ben-Or et al. [31] compile any interactive proof system into a *zero-knowledge* interactive proof system using cryptographic commitments. Finally, Bitansky et al. [47] compile a certain type of linear PCP into a succinct non-interactive argument of knowledge (SNARK) using either a "linear-only encryption" for the designated-verifier setting or a "linear-only one-way encoding," instantiated via bilinear groups, for the publicly verifiable setting. In this work we compile *fully linear* PCPs and IOPs into proofs on distributed or secret-shared data.

In the following we survey some of the information-theoretic proof systems used in prior work. For simplicity, we ignore the zero-knowledge feature that most of these systems have.

Let $\mathcal{L} \subseteq \{0,1\}^*$ be a language. Speaking informally, a proof system for $\mathcal{L}$ is a pair of (possibly interactive) algorithms (P, V). Both the prover P and verifier V take a string $x \in \{0,1\}^*$ as input (e.g., a SAT formula), and the prover's task is to convince the verifier that $x \in \mathcal{L}$ (e.g., that x is satisfiable). We sometimes view x as a vector over a finite field $\mathbb{F}$. We require the standard notions of *completeness* and *soundness*.

In the simplest such proof system, the prover sends the verifier a single proof string π of size $\text{poly}(|x|)$, the verifier reads x and π, and accepts or rejects. When the verifier is randomized and efficient, this setting corresponds to a Merlin-Arthur proof system [16]. There are a number of modifications to this basic paradigm that yield interesting alternative proof systems. In particular, we can:

- *Allow interaction between the prover and verifier.* In an interactive proof, the prover and verifier exchange many messages, after which the verifier must accept or reject. Allowing interaction may increase the power of the proof system [248] and makes it possible to provide zero-knowledge [151] in the plain model. (Alternatively, a common reference string is sufficient [51].)

- *Restrict the verifier's access to the proof.* Another way to modify the basic paradigm is to restrict the means by which the verifier interacts with the proof. In particular, we can view the proof as an oracle, and only allow the verifier to make a bounded (e.g., constant) number of queries to the proof oracle.

  In the classical PCP model [15, 125, 128], the proof is a string $\pi \in \Sigma^m$, for some finite alphabet Σ, and the verifier can only read a small number of symbols from the proof. On input i, the oracle returns the ith bit of the proof string π. (We call these "point queries.")

  In the linear PCP model [47, 170], the proof is a vector $\pi \in \mathbb{F}^m$, for some finite field $\mathbb{F}$, and the verifier can can only make a small number of "linear queries" to the proof. That is, the proof oracle takes as input a vector $q \in \mathbb{F}^m$ and returns the inner-product $\langle \pi, q \rangle \in \mathbb{F}$.

  In the polynomial PCP model [67], the proof is an ν-variate polynomial $\pi \in \mathbb{F}[Z_1, \ldots, Z_\nu]$, for some finite field $\mathbb{F}$. The verifier may evaluate this polynomial at a small number of points. That is, the proof oracle takes as input a point $q \in \mathbb{F}^\nu$ and outputs the evaluation $\pi(q) \in \mathbb{F}$. Typically, we think of the number of variables ν as being small relative to the degree of the polynomial, since otherwise the polynomial PCP and linear PCP models are equivalent.

- *Restrict the verifier's access to the input.* Yet another way to modify the basic paradigm is to restrict the verifier's access to the input x. In particular, we can view the *input* as an oracle, and only allow the verifier to make a bounded (e.g., constant) number of queries to the input oracle. We discuss the strong motivation for this model later on. We consider two variants.

  In a PCP of proximity [42], we view the input as a string and we only allow the verifier to make a limited number of point queries to the input string. With a few point queries, it is not possible to distinguish between an input $x \in \mathcal{L}$, and an input x "close to $\mathcal{L}$" (in Hamming distance). For this reason, PCPs of proximity necessarily provide only a relaxed notion of soundness: if x is "far from $\mathcal{L}$," then the verifier will likely reject.

  Alternatively, we can view the input as a vector $x \in \mathbb{F}^n$, for some finite field $\mathbb{F}$, and we only allow the verifier to make a small number of linear queries to the input x. That is, the input oracle takes as input a vector $q \in \mathbb{F}^n$ and returns the inner-product $\langle q, x \rangle \in \mathbb{F}$. We show that this notion, introduced and studied in this work, is sufficient to provide

	Proof type	Queries to input	Queries to proof	Representative compilers
Non-interactive	Classical proof (NP,MA) [16]	Read all	Read all	
	PCP [14, 15]	Read all	Point	Kilian [184], Micali [209]
	Linear PCP [170]	Read all	Linear	IKO [170], Pepper [246], GGPR [134], PHGR [225, 226], BCIOP [47]
	Polynomial PCP	Read all	Polynomial	
	PCP of proximity [42]	Point	Point	Kalai & Rothblum [178]
	Fully linear PCP	Linear	Linear	*This thesis*
Interactive	Interactive proof (IP) [151]	Read all	Read all	Ben Or et al. [31]
	IOP [39]	Read all	Point	BCS [39]
	Linear IOP	Read all	Linear	
	Polynomial IOP [67]	Read all	Polynomial	BFS [67], Sonic [205], Spartan [244]
	IOP of proximity [35, 36]	Point	Point	
	Fully linear IOP	Linear	Linear	*This thesis*, Hyrax [266], vSQL [281, 282]

Table 2.1: A comparison of information-theoretic proof systems. The **bolded** proof system models are ones that we introduce explicitly in this work. "Read all" refers to reading the entire data field, "Point" refers to reading a small number of cells of the data, and "Linear" refers to a making small number of linear queries to the data. "Polynomial" refers to viewing the proof as the coefficients of a (possibly multi-variate) polynomial and evaluating this polynomial at a small number of points.

a standard notion of soundness (unlike the relaxed notion of soundness that PCPs of proximity provide).

We now have three attributes by which we can classify information-theoretic proof systems: interactivity (yes/no), proof query type (read all/point/linear/polynomial), and input query type (read all/point/linear/polynomial). Taking the Cartesian product of these attributes yields 32 different possible proof systems, and we list 12 of particular interest in Table 2.1.

For example, interactive oracle proofs (IOPs) are interactive proofs in which the verifier has unrestricted access to the input but may make only point queries to proof strings [39]. Ben-Sasson et al. [39] show how to compile such proofs into succinct non-interactive arguments (SNARGs) in the random-oracle model. Recent work, including Ligero [8], STARK [34], and Aurora [38], constructs hash-based SNARGs using this technique.

Why fully linear proof systems? It is often the case that the verifier only has access to an additively homomorphic *encoding* of a statement x, and the prover convinces the verifier that the encoded statement is true. For example the verifier may be given an additively homomorphic commitment or encryption of the statement x. Or the verifier may

be implemented as a set of two or more servers who have a linear secret sharing of the statement x, or who hold different parts of x.

In all these settings, the verifiers can easily compute an *encoding* of the inner product of the statement x with a known query vector q. In some cases (such as the case of encrypted or committed data), the verifiers may need the prover's help help to "open" the resulting inner products.

When we compile fully linear PCPs into proof systems on shared, encrypted, or committed data, our compilers have the same structure: the prover sends an additively homomorphic encoding of the proof to the verifier. The verifier makes linear queries to the proof and input, and (if necessary) the prover provides "openings" of these linear queries to the verifier. The verifier checks that the openings are consistent with the encodings it was given, and then runs the fully linear PCP verifier to decide whether to accept or reject the proof.

The need for new constructions. In current applications of PCPs and linear PCPs, the length of the proof is not a complexity metric of much relevance. For example, in the BCIOP compiler [47] for compiling a linear PCP into a succinct non-interactive argument of knowledge (SNARK), the size of the proof corresponds to the prover's running time.

If the language $\mathcal{L}$ in question is decided by circuits of size $|C|$, then having proofs of size $|C|$ is acceptable, since the prover must run in time $\Omega(|C|)$ no matter what. A similar property holds for Micali's CS proofs [209], Kilian's PCP compiler [184], the BCS compiler [39] for interactive oracle proofs, and so on.

In our compilers, the prover must materialize the entire fully linear PCP proof, encode it, and send it to the verifier. For us, the size of the fully linear PCP proof not only dictates the running time of the prover, but also dictates the number of bits that the prover must communicate to the verifier. For this reason, in our setting, minimizing the proof size is an important goal.

Furthermore, when compiling linear PCPs into SNARKs using the existing compilers [47, 157, 226] it is critical that the linear PCP verifier have a concise representation as a degree-two arithmetic circuit. This is because the SNARK verifier runs the linear PCP verification "in the exponent" of a bilinear group. In contrast, our setting allows for more flexibility: the arithmetic degree of the verifier typically does not have a large impact on the cost of the compiled proof system.

Relating fully linear PCPs to streaming proof systems. The setting of *stream annotations* [76], introduced by Chakrabarti, Cormode, McGregor, and Thaler, restricts not only the verifier's access to the input and proof, but also the space usage of the verifier. In this model, the verifier is a space-bounded streaming algorithm: it may take a single pass over the input and proof, and must decide whether to accept or reject. For example, the verifier might be allowed only $O(\sqrt{n})$ bits of working space to decide inputs of length n. The *streaming interactive proof* model [92] is a generalization in which the prover and verifier may interact.

Fully linear interactive proofs naturally give rise to stream annotation proof systems. The reason is that if a fully linear PCP verifier makes q_π linear proof queries and q_x linear input queries, then the verifier can compute the responses to all of its queries by taking a single streaming pass over the input and proof while using $(q_x + q_\pi) \log_2 |\mathbb{F}|$ bits of space. Thus, fully linear PCPs with small proof size and query complexity give rise to stream annotation proof systems with small proof and space requirements. Similarly, fully linear IOPs give rise to streaming interactive proofs.

The implication in the other direction does not always hold, however, since stream annotation systems do not always give rise to fully linear PCPs with good parameters. The reason is that a streaming verifier may, in general, compute some non-linear function of the input that is difficult to simulate with linear queries.

Other proof systems. We briefly mention a number of other important classes of proof systems in the literature that are out of scope of this discussion. *Linear interactive proofs* are a model of interactive proof in which each message that the prover sends is an affine function of all of the verifier's previous messages, but is not necessarily an affine function of the input [47].

The fully linear PCP model is well matched to the problem of proving statements on data encoded with an additively homomorphic encoding, such as Paillier encryption [224] or a linear secret-sharing scheme. A different type of encoding is a *succinct* encoding, in which the prover can commit to a vector in $\mathbb{F}^m$ with a string of size sublinear in m [73, 180]. Bootle et al. [59] introduce the "Ideal Linear Commitment" (ILC) model as an abstraction better suited to this setting. In the ILC proof model, the prover sends the verifier *multiple* proofs vectors $\pi_1, \ldots, \pi_k \in \mathbb{F}^m$ in each round. The verifier is given a proof oracle that takes as input a vector $q \in \mathbb{F}^k$ and returns the linear combination $q^T \cdot (\pi_1 \ \ldots \ \pi_k) \in \mathbb{F}^m$. It is possible to translate linear IOP proofs into ILC proofs (and vice versa) up to some looseness in the parameters. A linear IOP in which the prover sends a length-m proof in each round implies

an ILC proof with the same query complexity in which the prover sends m proofs of length 1 in each round. An ILC proof in which the prover sends k proofs of length m and makes ℓ queries in each round implies a linear IOP with proof length $k \cdot m$ and query complexity $\ell \cdot m$. ILC-type proofs underlie the recent succinct zero-knowledge arguments of Bootle et al. [58] and Bünz et al. [66], whose security holds in the random-oracle model, assuming the hardness of the discrete-log problem.

Finally, another related notion from the literature is that of a *holographic proof* [17, 161], in which the verifier gets oracle access to an *encoding* of the input using an error-correcting code, typically a Reed-Muller code. Our notion of fully linear PCPs can be viewed as a variant of this model where the input is (implicitly) encoded by the Hadamard code and the proof can be accessed via *linear* queries, as opposed to point queries. In fact, our model allows a single linear query to apply *jointly* to the input and the proof.

We have not discussed multi-prover interactive proofs [32], in which multiple non-colluding provers interact with a single verifier, or more recently, multi-prover proofs in which a verifier gets access to multiple—possibly linear—proof oracles [56, 170].

2.2 Definitions

On concrete vs. asymptotic treatment. Since our new types of proof systems are meaningful objects even when all of the algorithms involved are computationally unbounded, our definitions refer to languages as finite objects and we do not explicitly track the running times of the various algorithms involved. All of our definitions natural extended to the standard asymptotic setting of infinite languages and relations with polynomial-time verifiers, honest provers, and simulators. Moreover, our constructions satisfy these asymptotic efficiency requirements. (See Remark 2.2.5 for details.)

2.2.1 Fully linear PCPs

Our new notion of *fully* linear PCPs build upon the definitions of standard linear PCPs from Ishai et al. [170] and Bitansky et al. [47]. We start by recalling the original notion.

Definition 2.2.1 (Linear PCP)**.** Let $\mathbb{F}$ be a finite field and let $\mathcal{L} \subseteq \mathbb{F}^n$ be a language. A linear probabilistically checkable proof system ("linear PCP") for $\mathcal{L}$ over $\mathbb{F}$ with proof

length m, soundness error ϵ, and query complexity ℓ is a pair of algorithms $(P_{\mathsf{FLPCP}}, V_{\mathsf{FLPCP}})$ with the following properties:

- For every $x \in \mathcal{L}$, the prover $P_{\mathsf{FLPCP}}(x)$ outputs a proof $\pi \in \mathbb{F}^m$.
- The verifier V_{FLPCP} consists of a query algorithm Q_{FLPCP} and a decision algorithm D_{FLPCP}. The query algorithm Q_{FLPCP} takes no input and outputs ℓ queries $q_1, \ldots, q_\ell \in \mathbb{F}^m$ and state information st. The decision algorithm D_{FLPCP} takes as input the instance x, the state st, and the ℓ answers $\langle q_1, \pi\rangle, \ldots, \langle q_\ell, \pi\rangle \in \mathbb{F}$ to Q_{FLPCP}'s queries. It outputs "accept" or "reject."

The algorithms additionally satisfy the following requirements:

- **Completeness.** For all $x \in \mathcal{L}$, the verifier accepts a valid proof:

$$\Pr\left[D_{\mathsf{FLPCP}}(\mathsf{st}, x, \langle q_1, \pi\rangle, \ldots, \langle q_\ell, \pi\rangle) = \text{"accept"} \ : \ \begin{array}{rl} \pi & \leftarrow P_{\mathsf{FLPCP}}(x) \\ (\mathsf{st}, q_1, \ldots, q_\ell) & \leftarrow Q_{\mathsf{FLPCP}}() \end{array}\right] = 1.$$

- **Soundness.** For all $x^* \notin \mathcal{L}$, and for all false proofs $\pi^* \in \mathbb{F}^m$, the probability that the verifier accepts is at most ϵ:

$$\Pr\left[D_{\mathsf{FLPCP}}(\mathsf{st}, x^*, \langle q_1, \pi^*\rangle, \ldots, \langle q_\ell, \pi^*\rangle) = \text{"accept"} \ : \ (\mathsf{st}, q_1, \ldots, q_\ell) \leftarrow Q_{\mathsf{FLPCP}}()\right] \leq \epsilon.$$

Remark 2.2.2. If we do not restrict the running time of the linear PCP verifier and we do not restrict the manner in which the verifier can access the statement x, then all computable languages have trivial linear PCPs: an inefficient linear PCP verifier can simply read x and test on its own whether $x \in \mathcal{L}$. To make the definition non-trivial, the standard notion of PCPs [254] (and also linear PCPs [47, 170]) restricts the verifier to run in time polynomial in the length of the input x. In contrast, a fully linear PCP—which we now define—restricts the *verifier's access to the statement* x by permitting the verifier to make a bounded number of linear queries to x. This restriction makes the definition non-trivial: even if the verifier can run in unbounded time, it cannot necessarily decide whether $x \in \mathcal{L}$ without the help of a proof π.

We now define the new notion of a *fully linear PCP* and its associated *strong zero knowledge* property.

Definition 2.2.3 (Fully linear PCP, "FLPCP")**.** We say that a linear PCP is *fully linear* if the decision predicate D_{FLPCP} makes only linear queries to both the statement x and to

the proof π. More formally, the query algorithm Q_{FLPCP} outputs queries $q_1, \ldots, q_\ell \in \mathbb{F}^{n+m}$, and state information st. The decision algorithm D_{FLPCP} takes as input the query answers $a_1 = \langle q_1, (x \| \pi) \rangle, \ldots, a_\ell = \langle q_\ell, (x \| \pi) \rangle$, along with the state st, and outputs an accept/reject bit.

Definition 2.2.4 (Degree of FLPCPs)**.** We say that a fully linear PCP has a degree-d verifier if:

- the state information st that the query algorithm outputs is in $\mathbb{F}^\mu$ and
- the decision algorithm D_{FLPCP} is computed by an arithmetic circuit of degree d. That is, there exists a test polynomial $T : \mathbb{F}^{\mu+\ell} \to \mathbb{F}^\eta$ of degree d such that $T(\mathsf{st}, a_1, \ldots, a_\ell) = 0^\eta$ if and only if $D_{\mathsf{FLPCP}}(\mathsf{st}, a_1, \ldots, a_\ell)$ accepts.

Remark 2.2.5 (Infinite languages). We can also define linear PCPs for infinite languages $\mathcal{L}$: for each $\lambda \in \mathbb{N}$, we have a relation $\mathcal{L}_\lambda$ over a field $\mathbb{F}(\lambda)$. We then we define $\mathcal{L} = \cup_{\lambda \in \mathbb{N}} \mathcal{L}_\lambda$. In this case, all of the algorithms that constitute the linear PCP also take as input the parameter λ written in unary. In addition, all parameters (n, m, ϵ, etc.) are functions of λ. (Alternatively, we can define an infinite language $\mathcal{L} \subseteq \mathbb{F}^*$, for some fixed finite field $\mathbb{F}$ and, for an input x, require that all algorithms run in time polynomial in $|x|$.) When considering linear PCPs for infinite languages $\mathcal{L} = \cup_{\lambda \in \mathbb{N}} \mathcal{L}_\lambda$, we can demand that the verifier V_{FLPCP} and simulator S_{FLPCP}, defined in the zero-knowledge property below, run in time polynomial in λ. In practice, we will also be interested in proof systems in which the honest prover P_{FLPCP} runs in polynomial time, provided that the language $\mathcal{L}$ is efficiently computable.

Zero knowledge. In our applications we will often need our linear PCPs to be zero-knowledge. We recall the standard notion of zero-knowledge for linear PCPs and introduce a strengthened formulation for the special case of fully linear PCPs.

Definition 2.2.6 (Zero-knowledge linear PCPs)**.** A linear PCP is *honest-verifier zero knowledge* ("HVZK") if there exists a simulator S_{FLPCP} such that for all $x \in \mathcal{L}$, the following distributions are identical:

$$S_{\mathsf{FLPCP}}(x) \equiv \left\{ \begin{array}{c} (\mathsf{st}, q_1, \ldots, q_\ell) \\ \big(\langle q_i, \pi \rangle, \ldots, \langle q_\ell, \pi \rangle\big) \end{array} : \begin{array}{rl} \pi & \leftarrow P_{\mathsf{FLPCP}}(x) \\ (\mathsf{st}, q_1, \ldots, q_\ell) & \leftarrow Q_{\mathsf{FLPCP}}() \end{array} \right\}.$$

Remark 2.2.7. In some applications, a relaxed version of the HVZK property is useful: we say that a linear PCP has δ-*statistical HVZK* if the two distributions defined in Definition 2.2.6

are not identical, but are δ-close in statistical distance

Definition 2.2.8 (Strong zero-knowledge fully linear PCPs)**.** A fully linear PCP is *strong honest-verifier zero knowledge* ("strong HVZK") if there exists a simulator S_{FLPCP} such that for all $x \in \mathcal{L}$, the following distributions are identical:

$$S_{\mathsf{FLPCP}}() \equiv \left\{ \begin{array}{c} (\mathsf{st}, q_1, \ldots, q_\ell) \\ \left(\langle q_1, (x\|\pi)\rangle, \ldots, \langle q_\ell, (x\|\pi)\rangle\right) \end{array} : \begin{array}{rl} \pi & \leftarrow P_{\mathsf{FLPCP}}(x, w) \\ (\mathsf{st}, q_1, \ldots, q_\ell) & \leftarrow Q_{\mathsf{FLPCP}}() \end{array} \right\}.$$

Remark 2.2.9. The strong zero-knowledge property here departs from the traditional zero-knowledge notion in that it essentially requires that an honest verifier learn *nothing* about the statement x by interacting with the prover, except that $x \in \mathcal{L}$ This notion is meaningful in our applications, since the statement x could be encrypted or secret-shared (for example), and thus it makes sense for a verifier to learn that $x \in \mathcal{L}$ without learning anything else about x.

2.2.2 Fully linear interactive oracle proofs

In a linear PCP, the interaction between the prover and verifier is "one-shot:" the prover produces a proof π, the verifier makes queries to the proof, and the verifier either accepts or rejects the proof. We define *fully linear interactive oracle proofs* ("fully linear IOPs"), generalizing linear PCPs to several communication rounds. This sort of linear proof system is inspired by the notion of IOPs from [39, 234] (generalizing an earlier notion of interactive PCPs [177]) that use point queries instead of linear queries. We define only the "public coin" variant of fully linear IOPs, since all of our constructions satisfy this notion.

In the ith round of a ρ-round fully linear IOP interaction, for $i \in \{1, \ldots, \rho\}$, the prover sends the verifier a proof $\pi_i \in \mathbb{F}^{m_i}$, where $\mathbb{F}$ is a finite field and m_i is a proof length parameter. The verifier then chooses a random challenge $r_i \xleftarrow{\mathrm{R}} \mathcal{S}_i$, where $\mathcal{S}_i$ is a finite set. The prover's next proof π_{i+1} may depend on the challenge r_i, and all of the messages it has seen so far. Finally, the verifier makes ℓ linear queries jointly to (1) the input and (2) the proofs it has seen so far. Based on the answers to these queries, the verifier accepts or rejects. The verifier's decision predicate is a function only of the verifier's public random challenges $(r_1, \ldots, r_\rho)$, the randomness used to generate the queries, and the answers to the verifier's queries $(q_1, \ldots, q_\ell)$.

A fully linear IOP must satisfy the natural notions of soundness, completeness, and honest-verifier zero knowledge.

Definition 2.2.10 (Public-coin fully linear interactive protocol)**.** A ρ-round ℓ-query public-coin fully linear interactive linear protocol Π with message length $(m_1, \ldots, m_\rho) \in \mathbb{N}^t$ and proof length $m = \sum_{i \in [\rho]} m_i$ over a finite field $\mathbb{F}$ consists of a prover algorithm P_{FLIOP} and a verifier $V_{\mathsf{FLIOP}} = (Q_{\mathsf{FLIOP}}, D_{\mathsf{FLIOP}})$, which consists of a query algorithm Q_{FLIOP} and a decision algorithm D_{FLIOP}. We define the output of the interaction $[P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}}](x)$ as the output of the following experiment:

$(\mathsf{st}_0^P, r_0) \leftarrow (x, \perp)$	// *Initialize prover state with input.*
For $i = 1, \ldots, \rho$:	// *In rounds* $1, 2, 3, \ldots, \rho$*:*
$\quad (\mathsf{st}_i^P, \pi_i) \leftarrow P_{\mathsf{FLIOP}}(\mathsf{st}_{i-1}^P, r_{i-1})$	// *Prover outputs ith proof.*
$\quad r_i \xleftarrow{\text{R}} \mathbb{F}^{b_i}$, for some parameter b_i	// *Verifier sends ith random challenge* r_i*.*
$(\mathsf{st}^V, q_1, \ldots, q_\ell) \leftarrow Q_{\mathsf{FLIOP}}(r_1, \ldots, r_\rho)$	// *Verifier generates queries.*
$\pi \leftarrow (\pi_1 \| \cdots \| \pi_\rho) \in \mathbb{F}^m$.	
$(a_1, \ldots, a_\ell) \leftarrow \big(\langle q_1, (x \| \pi)\rangle, \ldots, \langle q_\ell, (x \| \pi)\rangle\big)$	// *Answer queries.*
Output $D_{\mathsf{FLIOP}}\big(\mathsf{st}^V, a_1, \ldots, a_\ell\big)$	// *Verifier decides to accept/reject.*

On prover P_{FLIOP}, verifier $V_{\mathsf{FLIOP}} = (Q_{\mathsf{FLIOP}}, D_{\mathsf{FLIOP}})$, and input x, we denote the output of the above experiment as $[P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}}](x)$ and we say that the protocol accepts if the output of the experiment is "accept."

Definition 2.2.11 (Fully linear interactive oracle proof, FLIOP)**.** An interactive fully linear protocol $(P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}})$ is a *fully linear interactive oracle proof* system ("fully linear IOP") for a language $\mathcal{L}$ with soundness error ϵ if it satisfies the following properties:

- **Completeness.** For all $x \in \mathcal{L}$, the interaction $[P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}}](x)$ always accepts.
- **Soundness.** For all $x \notin \mathcal{L}$, and for all (computationally unbounded) P^*, $[P^*, V_{\mathsf{FLIOP}}](x)$ accepts with probability at most ϵ.

When the first round does not involve a proof but only a random challenge r_i, we deduct $1/2$ from the number of rounds. In particular, a 1.5-round public-coin fully linear IOP is one that involves, in this order:

- a random challenge r,
- a proof π, which may depend on r,
- queries $(q_1, \ldots, q_\ell)$ to $x \| \pi$, which may depend on fresh public randomness r', and
- a decision based on r, r' and the answers to the queries.

A fully linear IOP is *honest-verifier zero knowledge* if it additionally satisfies:

- **Honest-verifier zero knowledge.** There exists a simulator S_{FLIOP} such that

$$S_{\mathsf{FLIOP}}(x) \quad \equiv \quad \mathsf{View}_{[P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}}](x)}(V_{\mathsf{FLIOP}})$$

Linear PCP	Proof length	Queries	Verifier deg.	Soundness error
Hadamard LPCP [14, 47]	$O(\|C\|^2)$	3	2	$O(1)/\|\mathbb{F}\|$
GGPR-style [134]	$O(\|C\|)$	4	2	$O(\|C\|)/\|\mathbb{F}\|$
G-gates (Thm. 2.3.3)	$M \cdot \deg G$	$L+2$	$\deg G$	$M \cdot \deg G/(\|\mathbb{F}\| - M)$
Degree-two (Cor. 2.3.7)	$O(\sqrt{\|C\|})$	$O(\sqrt{\|C\|})$	2	$O(\sqrt{\|C\|})/\|\mathbb{F}\|$

Table 2.2: A comparison of existing and new fully linear PCP constructions for satisfiability of an arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}$. Proof length measures the number of field elements in $\mathbb{F}$. For the G-gates construction, $G : \mathbb{F}^L \to \mathbb{F}$ is an arithmetic circuit of total degree $\deg G$ and M is the number of G-gates in the circuit C.

for all $x \in \mathcal{L}$. Here, we use the notation $\mathsf{View}_{[P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}}](x)}(V_{\mathsf{FLIOP}})$ for the distribution of internal randomness and messages that V_{FLIOP} sees in its interaction with $P_{\mathsf{FLIOP}}(x)$. Furthermore, we say that a linear IOP satisfies *strong honest-verifier zero knowledge* ("strong HVZK") if the simulator S_{FLIOP} takes no input.

Definition 2.2.12 (Degree of fully linear IOP verifier)**.** We say that a fully linear PCP has a degree-d verifier if:

- the verifier's query routine outputs a state st^V that is a vector in $\mathbb{F}^\mu$, for some parameter $\mu \in \mathbb{N}$, and
- there exists a test polynomial $T : \mathbb{F}^{\mu+\ell} \to \mathbb{F}^\eta$ of degree d that takes as input (1) the verifier's query state and (2) the ℓ query responses to the verifier's queries at each round. The verifier accepts if and only if T evaluates to $0^\eta \in \mathbb{F}^\eta$.

2.3 Constructions: Fully linear PCPs

In this section we first show how to construct fully linear PCPs from existing linear PCPs. Next, we introduce a new fully linear PCP that yields shorter proofs for languages that are recognized by arithmetic circuits with certain repeated structure; the only cost is an increase in the algebraic degree of the verifier, which is irrelevant for our main applications. This new fully linear PCP is an important building-block for our new efficient fully linear IOP constructions in Section 2.4.

2.3.1 Constructions from existing linear PCPs

We begin by observing that the Hadamard [14, 47] and GGPR-style linear PCPs [37, 47, 134, 245], as described in the work of Bitansky et al. [47, Appendix A], satisfy our new notions of full linearity and strong zero knowledge.

Claim 2.3.1 (Informal)**.** *The Hadamard linear PCP and the GGPR-based linear PCP are constant-query* fully *linear PCPs, in the sense of Definition 2.2.3. Moreover, they yield fully linear PCPs with* strong *HVZK.*

The claim follows from a straightforward analysis of the Hadamard and GGPR-based linear PCPs. Since the linear PCP construction of Theorem 2.3.3 is fully linear and strong HVZK, and since it yields the GGPR-based linear PCP as a special case, we leave the claim unproven and instead focus on our new constructions.

2.3.2 The main theorem

We now describe a fully linear PCP for arithmetic circuit satisfiability, for circuits C with a certain type of repeated structure. (See Section 1.2 for a definition of arithmetic circuits.) When applied to arithmetic circuits of size $|C|$, it can yield proofs of length $o(|C|)$ field elements. In contrast, the existing general-purpose linear PCPs in Claim 2.3.1 have proof size $\Omega(|C|)$.

This new linear PCP construction applies to circuits that contain many instances of the same subcircuit, which we call a "G-gate." If the arithmetic degree of the G-gate is small, then the resulting linear PCP is short. More formally, we define:

Definition 2.3.2 (Arithmetic circuit with G-gates)**.** We say that a gate in an arithmetic circuit is an *affine gate* if (a) it is an addition gate, or (b) it is a multiplication gate in which one of the two input is a constant. Let $G : \mathbb{F}^L \to \mathbb{F}$ be an arithmetic circuit composed of affine gates and multiplication gates. An *arithmetic circuit with G-gates* is an arithmetic circuit composed of affine gates and G-gates.

The following theorem is the main result of this section. Recall that $|G|$ refers to the number of non-constant multiplication gates in the arithmetic circuit for G.

Theorem 2.3.3. *Let C be an arithmetic circuit with G-gates over $\mathbb{F}$ such that:*

(a) the gate $G : \mathbb{F}^L \to \mathbb{F}$ has total arithmetic degree $\deg G$*,*

(b) the circuit C consists of M instances of a G-gate and any number of affine gates, and

(c) the field $\mathbb{F}$ is such that $|\mathbb{F}| > M$.

Then, there exists a fully linear PCP with strong HVZK for the language $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid C(x) = 0\}$ that has:

- *proof length $L + M \deg G + 1$ elements of $\mathbb{F}$, where L is the arity of the G-gate,*
- *query complexity $L + 2$,*
- *soundness error $M \deg G/(|\mathbb{F}| - M)$, and*
- *a verification circuit of total degree $\deg G$ containing $|G|$ multiplication gates.*

Furthermore, if we require a fully linear PCP that is not necessarily strong HVZK, then the proof length decreases to $(M-1)\deg G + 1$ elements of $\mathbb{F}$ and the soundness error decreases to $M \deg G/|\mathbb{F}|$.

The proof of Theorem 2.3.3 uses the following simple fact about the linearity of polynomial interpolation and evaluation.

Fact 2.3.4. Let $\mathbb{F}$ be a finite field and let $\pi \in \mathbb{F}^m$. For some integer $n < |\mathbb{F}|$, let $A_1, \ldots, A_n$ be affine functions that map $\mathbb{F}^m$ to $\mathbb{F}$, and let $\alpha_1, \ldots, \alpha_n$ be distinct elements in $\mathbb{F}$. Define f to be the polynomial of lowest-degree such that $f(\alpha_i) = A_i(\pi)$ for all $i \in \{1, \ldots, n\}$. Then for all $r \in \mathbb{F}$ and all choices of the A_i, there exists a vector $\lambda_r \in \mathbb{F}^m$ and scalar $\delta_r \in \mathbb{F}$, such that $f(r) = \langle \lambda_r, \pi \rangle + \delta_r$ for all $\pi \in \mathbb{F}^m$.

Fact 2.3.4 says that given the values of a polynomial f at the points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ as affine functions of a vector $\pi \in \mathbb{F}^m$, we can express $f(r)$ as an affine function of π, and this affine function is independent of π. This follows from the fact that polynomial interpolation applied to the n points $\{(\alpha_i, A_i(\pi))\}_{i=1}^n$ followed by polynomial evaluation at the point r is an affine function of π.

Proof of Theorem 2.3.3. The construction that proves Theorem 2.3.3 is a generalization of the linear PCP implicit in the construction used in the Prio system [93] and is closely related to a Merlin-Arthur proof system of Williams for batch verification of circuit evaluation [272]. Figure 2.1 gives an example of the proof construction, applied to a particular simple circuit.

Let $\alpha_0, \alpha_1, \ldots, \alpha_M \in \mathbb{F}$ be $M + 1$ fixed distinct elements in the field $\mathbb{F}$. Such elements exist because the hypothesis of the theorem asserts that $|\mathbb{F}| > M$.

Label the G-gates of the circuit C in topological order from inputs to outputs; there are M such gates in the circuit. Without loss of generality, we assume that the output of the

circuit C is the value on the output wire of the last G-gate in the circuit.

FLPCP prover. On input $x \in \mathbb{F}^n$, the prover evaluates the circuit $C(\cdot)$ on the input x. The prover then defines L polynomials $f_1, \ldots, f_L \in \mathbb{F}[X]$ such that, for every $i \in \{1, \ldots, L\}$,

(i) the term $f_i(\alpha_0)$ is a value chosen independently and uniformly at random from $\mathbb{F}$, and

(ii) for all $j \in \{1, \ldots, M\}$, $f_i(\alpha_j) \in \mathbb{F}$ is the value on the i-th input wire to the j-th G-gate when evaluating the circuit C on the input x.

Furthermore, the prover lets $f_1, \ldots, f_L$ be the polynomials of lowest degree that satisfy these relations. Observe that each of the polynomials $f_1, \ldots, f_L$ has degree at most M.

Next, the prover constructs a proof polynomial $p = G(f_1, \ldots, f_L) \in \mathbb{F}[X]$. By construction of p, we know that, for $j \in \{1, \ldots, M\}$, $p(\alpha_j)$ is the value on the output wire from the j-th G-gate in the evaluation of $C(x)$. Moreover, $p(M) = C(x)$. Let d be the degree of the polynomial p and let $c_p \in \mathbb{F}^{d+1}$ be the vector of coeffcients of $p \in \mathbb{F}[X]$. By construction, the degree of p satisfies $d \leq M \deg G$.

The prover outputs $\pi = (f_1(\alpha_0), \ldots, f_L(\alpha_0),\ c_p) \in \mathbb{F}^{L+d+1}$ as the linear PCP proof.

(*Note:* If we do not require strong HVZK to hold, then the prover need not randomize the constant terms of the polynomials $f_1, \ldots, f_L$. In this case, the prover does not include the values $f_1(0), \ldots, f_L(0)$ in the proof, and the degree of the polynomial p decreases to $(M-1)\deg G$. Thus, if we do not require strong HVZK, the proof length falls to $(M-1)\deg G + 1$.)

FLPCP queries. We can parse the (possibly maliciously crafted) proof $\pi \in \mathbb{F}^{L+d+1}$ as: the values $(z'_1, \ldots, z'_L) \in \mathbb{F}^L$ representing the values of some polynomials $f'_1, \ldots, f'_L$ evaluated at the point $\alpha_0 \in \mathbb{F}$, and the coefficients $c'_p \in \mathbb{F}^{d+1}$ of a polynomial $p' \in \mathbb{F}[X]$ of degree at most d. If the proof is well-formed, the polynomial p' is such that $p'(\alpha_j)$ encodes the output wire of the jth G-gate in the circuit $C(\cdot)$ when evaluated on the input x.

Given p', we define L polynomials $f'_1, \ldots, f'_L \in \mathbb{F}[X]$ such that:

(i) the polynomial satisfies $f'_i(\alpha_0) = z'_i$, where z'_i is the value included in the proof π', and

(ii) $f'_i(\alpha_j) \in \mathbb{F}$ is the value on the i-th input wire to the j-th G-gate in the circuit, under the purported assignment of values to the output wires of the G-gates implied by the polynomial p' and witness w'.

More precisely, we define $f'_i(\alpha_j)$ inductively: The value on the ith input wire to the jth G-gate in the circuit $C(x)$ is some affine function A_{ij} of

- the input $x \in \mathbb{F}^n$ and
- the purported outputs of the first $j-1$ G-gates in the circuit: $p'(\alpha_1), \ldots, p'(\alpha_{j-1}) \in \mathbb{F}$.

So, for all $i \in \{1, \ldots, L\}$, we define f'_i to be the polynomial of least degree satisfying:

$$f'_i(\alpha_0) = z'_i$$
$$f'_i(\alpha_j) = A_{ij}(x, p'(\alpha_1), ..., p'(\alpha_{j-1})) \qquad \text{for} \quad 1 \leq j \leq M,$$

where A_{ij} is a fixed affine function defined by the circuit C.

The verifier's goal is to check that:

1. $p' = G(f'_1, \ldots, f'_L)$, and,
2. the circuit output $p'(\alpha_M)$ satisfies $p'(\alpha_M) = 0$.

As we argue below, the first condition ensures that $p'(M)$ is equal to the output of the circuit $C(x, w')$. The second check ensures that the output is 0.

To implement the first check, the verifier samples a random point $r \xleftarrow{R} \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$ and outputs query vectors that allow evaluating p' and $f'_1, \ldots, f'_L$ at the point r. (For the honest-verifier zero knowledge property to hold, it is important that we exclude the set $\{\alpha_1, \ldots, \alpha_M\}$ from the set of choices for r.) The verifier has linear access to the input x and terms $z' = (z'_1, \ldots, z'_L)$, and the coefficients $c'_p \in \mathbb{F}^{d+1}$ of the polynomial p'. Hence, using Fact 2.3.4, it follows that the query algorithm can compute vectors $\lambda_1, \ldots, \lambda_L \in \mathbb{F}^{n+L+d+1}$ and scalars $\delta_1, \ldots, \delta_L \in \mathbb{F}$ such that $f'_i(r) = \langle \lambda_i,\ (x \| z' \| c'_p) \rangle + \delta_i$ for $i \in \{1, \ldots, L\}$, where $r \in \mathbb{F}$ is the random point chosen above. Similarly, the query algorithm can compute a vector $\lambda \in \mathbb{F}^{n+L+d+1}$ such that $p'(r) = \langle \lambda,\ (x \| z' \| c'_p) \rangle$.

The verifier can execute the second check, to ensure that $p'(\alpha_M) = 0$, with a single linear query.

FLPCP decision. The decision algorithm takes as input the state value $r \in \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$, along with the query answers $a, a_1, \ldots, a_L, b \in \mathbb{F}$, where $a = p'(r)$, $a_i = f'_i(r)$ for $i \in \{1, \ldots, \ell\}$, and $b = p'(\alpha_M)$. The verifier accepts if $a = G(a_1, \ldots, a_L)$ and $b = 0$.

Security arguments. We show completeness, soundness, and strong HVZK.

Completeness. If the prover is honest, then $p' = G(f'_1, \ldots, f'_L)$ and $p'(\alpha_M) = 0$ by construction. The verifier will always accept in this case.

Soundness. Fix a circuit C, an input $x \in \mathbb{F}^n$, and a proof $\pi' \in \mathbb{F}^{L+d+1}$. We show that if $x \notin \mathcal{L}$ then the verifier accepts with probability at most $M \deg G/(|\mathbb{F}| - M)$.

As in the description of the query algorithm, we can view:

- the first L elements of the proof as terms $z'_1, \ldots, z'_L \in \mathbb{F}$, and
- the latter elements as the coefficients of a polynomial p' of degree at most $d \leq M \deg G$.

We may assume that $p'(\alpha_M) = 0$, since otherwise the verifier always rejects. In the discussion that follows, let the polynomials $f'_1, \ldots, f'_L$ be the ones defined in the description of the linear PCP query algorithm.

We claim that if for all $j \in \{1, \ldots, M\}$, it holds that $p'(\alpha_j) = G(f'_1(\alpha_j), \ldots, f'_L(\alpha_j))$, then for all $j \in \{1, \ldots, M\}$, $p'(\alpha_j)$ encodes the value of the output wire of the jth G-gate in the circuit C when evaluated on input x.

We prove this claim by induction on j:

- *Base case ($j = 1$).* The values $(f'_1(\alpha_1), \ldots, f'_L(\alpha_1))$ depend only on the input x. By construction, the values $(f'_1(\alpha_1), \ldots, f'_L(\alpha_1))$ are exactly the values of the input wires to the first G-gate in the evaluation of $C(x)$. Then if $p'(\alpha_1) = G(f'_1(\alpha_1), \ldots, f'_L(\alpha_1))$, $p'(\alpha_1)$ encodes the value on the output wire of the first G-gate.
- *Induction step.* Assume that, for all $k \in \{1, \ldots, j-1\}$, $p'(\alpha_k) = G(f'_1(\alpha_k), \ldots, f'_L(\alpha_k))$. Then, by the induction hypothesis, $(p'(\alpha_1), \ldots, p'(\alpha_{j-1}))$ are the values on the output wires of the first $j-1$ G-gates of C, when evaluated on input x.

  The values $(f'_1(\alpha_j), \ldots, f'_L(\alpha_j))$ are affine functions of x and the values $p'(\alpha_1), \ldots, p'(\alpha_{j-1})$. Then, by construction of the polynomials $(f'_1, \ldots, f'_L)$, the values $(f'_1(\alpha_j), \ldots, f'_L(\alpha_j))$ encode the values on the input wires to the j-th G-gate in the evaluation of the circuit $C(x)$. Finally, if we assume that $p'(\alpha_j) = G(f'_1(\alpha_j), \ldots, f'_L(\alpha_j))$, then $p'(\alpha_j)$ must be the value on the output wire of the jth G-gate.

  We have thus proved the induction step.

This completes the proof of the claim.

If $p'(\alpha_M) = 0$ (as we have assumed), but $x \notin \mathcal{L}$, then $p'(\alpha_M)$ does not encode the output value of the Mth G-gate in the evaluation of the circuit $C(x)$. By the claim just proved, this implies that for some $j^* \in \{1, \ldots, M\}$, $p'(\alpha_{j^*}) \neq G(f'_1(\alpha_{j^*}), \ldots, f'_L(\alpha_{j^*}))$. Thus, when we view $p', f'_1, \ldots, f'_L \in \mathbb{F}[X]$ as univariate polynomials, we have that $p' \neq G(f'_1, \ldots, f'_L)$.

Now, if $p' \neq G(f'_1, \ldots, f'_L)$ then $p' - G(f'_1, \ldots, f'_L) \in \mathbb{F}[X]$ is a non-zero univariate polynomial of total degree at most $M \deg G$. Such a polynomial can have at most $M \deg G$ roots

over $\mathbb{F}$. Therefore the probability, over the verifier's random choice of $r \xleftarrow{R} \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$, that $p'(r) - G(f'_1(r), \ldots, f'_L(r)) = 0$ is at most $M \deg G/(|\mathbb{F}| - M)$. We conclude that the verifier accepts a false proof with probability at most $M \deg G/(|\mathbb{F}| - M)$.

Strong honest-verifier zero knowledge. To show that the construction satisfies strong HVZK, we must produce a simulator $S_{\mathsf{FLPCP}}()$ that perfectly simulates the joint distribution of the honest verifier's queries and the honest prover's responses. The honest verifier's queries are determined by the random choice of the point $r \xleftarrow{R} \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$ at which the verifier evaluates the polynomials $p, f_1, \ldots, f_L$. The simulator must then simulate the distribution of values $\langle r,\ p(r), f_1(r), \ldots, f_L(r),\ p(\alpha_M)\rangle \in \mathbb{F}^{L+3}$.

The simulator S_{FLPCP} takes no input and executes the following steps:

- Choose $r \xleftarrow{R} \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$.
- Choose $a_1, \ldots, a_L \xleftarrow{R} \mathbb{F}$.
- Compute $a \leftarrow G(a_1, \ldots, a_L) \in \mathbb{F}$.
- Output the tuple $\langle r,\ a, a_1, \ldots, a_L,\ 0\rangle \in \mathbb{F}^{L+3}$.

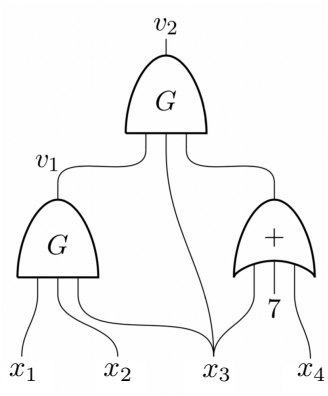
The simulated value r is distributed exactly as in the real interaction. Since $p(\alpha_M) = 0$ in the real interaction, the simulation of this value is also perfect. If the simulation $(a_1, \ldots, a_L)$ of the values $(f_1(r), \ldots, f_L(r))$ is perfect, then the simulation a of the value $p(r)$ is also perfect, since a is constructed exactly as in the real interaction.

We must then only argue that the simulation $(a_1, \ldots, a_L)$ of the values $(f_1(r), \ldots, f_L(r))$ is correct. For every $i \in \{1, \ldots, L\}$, we can write the value $f_i(r)$ in terms of the Lagrange interpolating polynomials $\lambda_0(\cdot), \lambda_1(\cdot), \ldots, \lambda_M(\cdot)$, evaluated at the point r:

$$f_i(r) = \lambda_0(r) \cdot f_i(0) + \sum_{j \in [M]} \lambda_j(r) \cdot f_i(\alpha_j).$$

When $r \notin \{\alpha_1, \ldots, \alpha_M\}$, the value of the zero-th interpolating polynomial is non-zero: $\lambda_0(r) \neq 0$. Since, by construction, the value $f_i(\alpha_0)$ is distributed uniformly at random over $\mathbb{F}$ and is independent of all other values, when $r \notin \{\alpha_1, \ldots, \alpha_M\}$, $f_i(r)$ will be distributed uniformly over $\mathbb{F}$ and independently of all other values.

Since the honest verifier chooses $r \xleftarrow{R} \mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}$, we conclude that the joint distribution of $(r, f_1(r), \ldots, f_L(r))$ will be uniform over $(\mathbb{F} \setminus \{\alpha_1, \ldots, \alpha_M\}) \times \mathbb{F}^L$ in the real interaction. The entire simulation is then perfect. □



Circuit. An example circuit C using an arbitrary three-input G-gate. The circuit takes as input the vector $(x_1, x_2, x_3, x_4) \in \mathbb{F}^4$. The circuit C outputs v_2, the value on the output wire of the topologically last G-gate.

Linear PCP proof. Using Theorem 2.3.3, we construct a fully linear PCP proof π that the input $(x_1, x_2, x_3, x_4) \in \mathbb{F}^3$ is in the language recognized by C. That is, the prover asserts that $C(x_1, x_2, x_3, x_4) = 0 \in \mathbb{F}$.

For this example, we assume that $\mathbb{F}$ has large characteristic so that we can identify the values $(\alpha_0, \alpha_1, \ldots, \alpha_d) \in \mathbb{F}$ used in the linear PCP construction with the integers $(0, 1, 2, \ldots, d)$.

The prover first constructs three polynomials f_1, f_2, f_3. The value $f_i(j)$ encodes the value on the i-th input to the j-th G-gate, in topological order from inputs to outputs. The constant terms are random elements $z_1, z_2, z_3 \xleftarrow{R} \mathbb{F}$. That is:

$f_1(0)=z_1$	$f_2(0)=z_2$	$f_3(0)=z_3$
$f_1(1)=x_1$	$f_2(1)=x_2$	$f_3(1)=x_3$
$f_1(2)=v_1=G(x_1,x_2,x_3)$	$f_2(2)=x_3$	$f_3(2)=x_3+x_4+7$

Next, the prover constructs the polynomial p, which satisfies $p = G(f_1, f_2, f_3)$, and which has degree at most $d = 2 \deg G$. Notice that for $j \in \{1, 2\}$, $p(j)$ is the value on the output wire of the j-th G-gate. Letting $d = 3 \deg G$, we can write the values of p as:

$$
\begin{aligned}
&p(0)=G(f_1(0),f_2(0),f_3(0))=G(z_1,z_2,z_3)\\
&p(1)=G(f_1(1),f_2(1),f_3(1))=v_1=G(x_1,x_2,x_3)\\
&p(2)=G(f_1(2),f_2(2),f_3(2))=v_2=G(v_1,x_3,x_3+x_4+7)\\
&p(3)=G(f_1(3),f_2(3),f_3(3))\\
&\quad\vdots\\
&p(d)=G(f_1(d),f_2(d),f_3(d))
\end{aligned}
$$

The linear PCP proof π consists of the elements: $(z_1, z_2, z_3, \bar{p}) \in \mathbb{F}^{L+d+1}$, where $\bar{p} \in \mathbb{F}^{d+1}$ is the vector of coefficients of the polynomial p.

Figure 2.1: An example of the fully linear PCP proof of Theorem 2.3.3.

If we define the G-gate to be a multiplication gate, so that $\deg G = 2$, then the construction of Theorem 2.3.3 matches the complexity of the GGPR-based linear PCP [134, 245] and provides what is essentially an alternative formulation of that proof system. In contrast, if $\deg G \ll |G|$, then this construction can yield significantly shorter proofs than the GGPR-based linear PCP, at the cost of increasing the algebraic degree of the verifier from 2 to $\deg G$.

Remark 2.3.5. We can generalize Theorem 2.3.3 to handle circuits with many distinct repeated subcircuits $G_1, \ldots, G_q$ with M_i instances of each gate $G_i : \mathbb{F}^{L_i} \to \mathbb{F}$, for $i \in \{1, \ldots, q\}$. The resulting fully linear PCP with strong HVZK has proof length at most $(\sum_{i\in[q]} L_i) + (\sum_{i\in[q]} M_i \deg G_i) + q$ elements of $\mathbb{F}$, query complexity $1 + \sum_{i\in[q]}(L_i + 1)$, a verifier of algebraic degree $\max_i \deg G_i$, and soundness error $\sum_{i\in[q]} \big(M_i \deg G_i/(|\mathbb{F}| - M_i)\big)$.

Remark 2.3.6. To get good soundness when applying the proof system of Theorem 2.3.3, the field $\mathbb{F}$ must be such that $|\mathbb{F}| \gg M \deg G + M$. In many applications, the input $x \in \mathbb{F}^n$ is a vector in a small field, such as the binary field $\mathbb{F}_2$. In this case, we apply Theorem 2.3.3 by lifting x into an extension field $\widetilde{\mathbb{F}}$ of $\mathbb{F}$, and carrying out the linear PCP operations in the extension.

The randomization technique we use to achieve honest-verifier zero-knowledge in Theorem 2.3.3 is inspired by the one that appears in the work of Bitansky et al. [47] for achieving HVZK in the Hadamard linear PCP construction.

2.3.3 Application: Short proofs for degree-two languages

As an application of Theorem 2.3.3 we demonstrate a special-purpose fully linear PCP for relations recognized by arithmetic circuits of degree two. When applied to an arithmetic circuit $C : \mathbb{F}^n \to \mathbb{F}$, we obtain a proof that consists of only $O(\sqrt{n})$ field elements and whose query complexity is only $O(\sqrt{n})$. For general-purpose linear PCPs, such as the Hadamard or GGPR-based linear PCPs, the proof length plus query complexity is much larger: $\Omega(n)$.

A special case of this proof yields a linear PCP for the language of vectors whose inner product is equal to a certain value. To give one application of such a proof system: Given encryptions of two sets, represented by their characteristic vectors, this proof system would allow a prover to succinctly show that the sets are disjoint.

This construction also reveals the close connection between fully linear PCPs and communication complexity. Without zero knowledge, this proof protocol boils down to the

Merlin-Arthur communication complexity protocol of Aaronson and Wigderson [1]. Furthermore, as we show in Section 2.5, we can use lower bounds on the communication complexity of inner-product to show that this fully linear PCP construction has essentially optimal parameters.

Corollary 2.3.7 (FLPCP for degree-two circuits)**.** *Let $\mathbb{F}$ be a finite field, let $C : \mathbb{F}^n \to \mathbb{F}$ be an arithmetic circuit of degree two (i.e., a degree-two polynomial over $\mathbb{F}$), and let $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid C(x) = 0\}$. There is a fully linear PCP with strong HVZK for $\mathcal{L}_C$ that has proof length $O(\sqrt{n})$ elements of $\mathbb{F}$, query complexity $O(\sqrt{n})$, a verifier of algebraic degree two, and soundness error $\frac{O(\sqrt{n})}{|\mathbb{F}|-\lceil\sqrt{n}\rceil}$.*

The idea of Corollary 2.3.7 is that any degree-two circuit $C : \mathbb{F}^n \to \mathbb{F}$ can be expressed as a circuit that computes an inner-product of dimension-n vectors, along with some number of affine gates. This property is special to degree-two circuits—the idea does not immediately generalize to circuits of higher constant degree.

Proof of Corollary 2.3.7. Without loss of generality we can assume that C implements a quadratic form $C(x) = x^T \cdot A \cdot x$ for some matrix $A \in \mathbb{F}^{n \times n}$. Indeed, a proof system for quadratic forms yields a proof system for any circuit of degree two. We can re-write $C(x)$ as the inner-product of the vectors x and $z = A \cdot x$ in $\mathbb{F}^n$. Hence, it suffices to design a fully linear PCP for the inner-product language $\mathcal{L}'_C = \left\{x \in \mathbb{F}^n \;\middle|\; \langle x \;,\; A \cdot x\rangle = 0\right\}$.

Let L^2 be the closest perfect square greater than or equal to n, and pad the vectors x and $z = A \cdot x$ with zeros so that both are in $\mathbb{F}^{(L^2)}$. Next, arrange the vector x into a matrix $X \in \mathbb{F}^{L \times L}$, and arrange z into a matrix $Z \in \mathbb{F}^{L \times L}$ in the same way. Then $C(x) = \langle x, z\rangle = \text{trace}(X \cdot Z^T)$.

Because the trace is a linear function, we can compute $C(x)$ using a circuit C' consisting of only addition gates and a total of L gates $G : \mathbb{F}^L \times \mathbb{F}^L \to \mathbb{F}$ defined as $G(u, v) = \langle u, v\rangle$ for $u, v \in \mathbb{F}^L$. It holds that $\deg G = 2$ and $L = O(\sqrt{n})$. Applying Theorem 2.3.3 to this G-gate circuit gives a fully linear PCP for $\mathcal{L}_{C'}$ with strong HVZK with the parameters stated in the corollary, as required. The proof needs one additional linear query to verify that the padding in x and z is all zero, but this does not change the parameters in the corollary. □

2.3.4 Application: Short proofs for parallel-sum circuits

As a second application of Theorem 2.3.3, we give a special-purpose fully linear PCP for languages recognized by circuits that take as input a vector $x \in \mathbb{F}^n$ and:

- apply an affine transformation to the input,
- apply the same sub-circuit $C : \mathbb{F}^L \to \mathbb{F}$ in parallel to each block of L values, and
- sum the outputs of the C circuits.

More formally, let $C : \mathbb{F}^L \to \mathbb{F}$ be an arithmetic circuit. Let $A : \mathbb{F}^n \to \mathbb{F}$ and $A_1, \ldots, A_M : \mathbb{F}^n \to \mathbb{F}^L$ be affine functions. This linear PCP construction applies to the language of values $x \in \mathbb{F}^n$ such that $\sum_{i \in [M]} C(A_i(x)) = A(x)$.

Corollary 2.3.8 (FLPCP for parallel-sum circuits)**.** *Let $C : \mathbb{F}^L \to \mathbb{F}$ be an arithmetic circuit over $\mathbb{F}$ that has arithmetic degree $\deg C$. Let $A : \mathbb{F}^n \to \mathbb{F}$ and $A_1, \ldots, A_M \in \mathbb{F}^n \to \mathbb{F}^L$ be affine functions. Then, there exists a strong HVZK fully linear PCP for the language $\mathcal{L}_{C,A,A_1,\ldots,A_M} = \{x \in \mathbb{F}^n \mid \sum_{i \in [M]} C(A_i(x)) = A(x)\}$ that has:*

- *proof length $O(\sqrt{M} \cdot (L + \deg C))$ elements of $\mathbb{F}$,*
- *query complexity $O(\sqrt{M} \cdot L)$,*
- *soundness error $\frac{\sqrt{M} \cdot \deg C}{|\mathbb{F}| - \sqrt{M}}$, and*
- *an arithmetic verification circuit of degree $\deg C$ containing $O(\sqrt{M} \cdot |C|)$ multiplication gates.*

Proof of Corollary 2.3.8. We define an appropriate G-gate and then invoke Theorem 2.3.3. Assume that M is a perfect square, since otherwise we can pad M up to the nearest square. The gadget $G : \mathbb{F}^{\sqrt{M}L} \to \mathbb{F}$ applies the circuit C to $\sqrt{M}$ blocks of L inputs. So, on input $(\bar{x}_1, \ldots, \bar{x}_{\sqrt{M}}) \in \mathbb{F}^{\sqrt{M}L}$, where $\bar{x}_j \in \mathbb{F}^L$ for all $j \in \{1, \ldots, \sqrt{M}\}$, the G-gate outputs:

$$G(\bar{x}_1, \ldots, \bar{x}_{\sqrt{M}}) \stackrel{\text{def}}{=} \sum_{j \in [\sqrt{M}]} C(\bar{x}_j) \quad \in \mathbb{F}. \tag{2.1}$$

Then the language $\mathcal{L}_{C,A,A_1,\ldots,A_M}$ is recognized by a circuit containing $M' = \sqrt{M}$ instances of the G-gate, along with some number of affine gates. Applying Theorem 2.3.3 using this G-gate yields a fully linear PCP with the desired efficiency properties. □

Example 2.3.9. Corollary 2.3.8 gives a short fully linear PCP for proving that a vector $x \in \mathbb{F}^n$ has L_p-norm y, for any integer $p \geq 2$. For this application, $C(x_i) \stackrel{\text{def}}{=} x_i^p \in \mathbb{F}$ and $\deg C = p$. The linear PCP implied by the corollary has length $O(p \cdot \sqrt{n})$, whereas the standard linear PCPs from Section 2.3.1 would have proofs of length $\Omega(n \log p)$. Thus, the construction of the corollary has shorter proof size for all $p = o(\sqrt{n})$.

A sum-check-style protocol, along the lines of Cormode, Thaler, and Yi's protocol [92, Section 3.2] for checking the k-th frequency moment in a data stream, achieves the same asymptotic complexity as the protocol of Example 2.3.9. However, that sum-check protocol does not natively provide zero knowledge. We thank Justin Thaler for pointing out this alternative construction.

2.4 Constructions: Fully linear interactive oracle proofs

In this section, we construct fully linear interactive oracle proofs ("fully linear IOPs"). While fully linear IOPs in general require multiple rounds of interaction between the prover and verifier, this extra interaction can sharply decrease the total proof length and verifier time.

For example, we give an $O(\log n)$-round fully linear IOP for proving that a vector $x \in \mathbb{F}^n$ consists entirely of 0/1 entries, where the total proof size consists only of $O(\log n)$ field elements. (See Example 2.4.7.) In comparison, linear PCPs for this language yield proofs of size $\Omega(n)$.

Several protocols from the literature, including the "Muggles" scheme of Goldwasser, Kalai, and Rothblum [149, 148], implicitly construct fully linear IOPs. We describe the connection between our notion and these protocols in Section 2.4.4.

2.4.1 A recursive fully linear IOP for parallel-sum circuits

Corollary 2.3.8 gives a linear PCP for "parallel-sum" circuits whose length grows as the square root of the degree of parallelism. Here, we show that by increasing the number of rounds of interaction between the prover and verifier, we can decrease the proof size to *logarithmic* in the degree of parallelism. The key observation is that in Corollary 2.3.8, the linear PCP verifier is itself a parallel-sum circuit. So rather than having the verifier evaluate this circuit on its own, the verifier can outsource the work of evaluating the verification circuit to the prover. The prover then uses a secondary linear PCP to convince the verifier that it executed this step correctly.

To get the optimal bounds we rebalance the parameters used in the proof of Corollary 2.3.8. Instead of a G-gate containing $\sqrt{M}$ copies of C, as in (2.1), we use a G-gate containing $M/2$ copies of C, and then recursively verify one input/output pair for that G-gate.

Theorem 2.4.1. *Let $C : \mathbb{F}^L \to \mathbb{F}$ be an arithmetic circuit over $\mathbb{F}$ that has arithmetic degree $\deg C$. Let $A : \mathbb{F}^n \to \mathbb{F}$ and $A_1, \ldots, A_M : \mathbb{F}^n \to \mathbb{F}^L$ be affine functions. Then, there exists an*

$O(\log M)$-round strong HVZK fully linear IOP for the language $\mathcal{L}_{C,A,A_1,\ldots,A_M} = \{x \in \mathbb{F}^n \mid \sum_{i\in[M]} C(A_i(x)) = A(x)\}$ that has:

- *proof length $L + O(\deg C \cdot \log M)$ elements of $\mathbb{F}$,*
- *query complexity $L + O(\log M)$,*
- *soundness error $O\left(\frac{\deg C \cdot \log M}{|\mathbb{F}|}\right)$, and*
- *an arithmetic verification circuit containing $O(|C|)$ multiplication gates.*

Figure 2.2 gives a graphical depiction of the protocol flow.

Proof of Theorem 2.4.1. We prove the theorem by induction on M. For simplicity, we assume that M is a power of two.

Base case ($M = 1$). When $M = 1$, we can invoke Theorem 2.3.3 using the circuit C as the G-gate of the theorem. Theorem 2.3.3 implies that there exists a one-round fully linear IOP with strong HVZK that has proof length $L + \deg C + 1$, query complexity $L + 2$, soundness error $\deg C/(|\mathbb{F}| - 1)$, and a verification circuit containing $|C|$ multiplication gates. This proves the base case.

Induction step. We define a G-gate, as in (2.1). The difference is that we now use a G-gate whose width is $M/2$ rather than $\sqrt{M}$. Specifically, using the notation in (2.1), we define G as:

$$G(\bar{x}_1, \ldots, \bar{x}_{M/2}) \stackrel{\text{def}}{=} \sum_{i\in[M/2]} C(\bar{x}_i) \quad \in \mathbb{F}.$$

The language $\mathcal{L}_{C,A,A_1,\ldots,A_M}$ is recognized by a circuit composed of two such G-gates, along with some number of affine gates.

The $O(\log M)$-round protocol then proceeds as follows:

- The prover sends a fully linear PCP proof π asserting that $x \in \mathbb{F}^n$ is in $\mathcal{L}_{C,A,A_1,\ldots,A_M}$. That is, the proof asserts that x satisfies the relation $\sum_{i\in[M]} C(A_i(x)) = A(x)$. To generate the proof, the prover uses the construction of Theorem 2.3.3 *without* strong HVZK. When invoking Theorem 2.3.3, the prover uses a G-gate as in (2.1), except that it has width $M/2$.

  The proof has length $2 \deg G + 1 = 2 \deg C + 1$, query complexity $(M/2)L + 2$, soundness error $2 \deg C/(|\mathbb{F}| - 2)$, and requires the verifier to evaluate a circuit of size $|G| = (M/2) \cdot |C|$ on the query answers.

Prover		Verifier
1a. Generate a linear PCP proof π_1 that $x \in \mathcal{L}_1$.	$\xrightarrow{\pi_1}$	
	$\xleftarrow{r_1}$	1b. Sample a random value $r_1 \xleftarrow{R} \mathbb{F}$ and send it to the prover. Make one linear query to the proof.
2a. Generate a proof π_2 that $(x\|\pi_1) \in \mathcal{L}_2$, where $\mathcal{L}_2$ is the language of vectors $(x\|\pi)$ such that $V_{\mathcal{L}}(x\|\pi_1; r_1) = 1$.	$\xrightarrow{\pi_2}$	
	$\xleftarrow{r_2}$	2b. Sample a random value $r_2 \xleftarrow{R} \mathbb{F}$ and send it to the prover. Make one linear query to the proof.
3a. Generate a proof π_3 that $(x\|\pi_1\|\pi_2) \in \mathcal{L}_3$, where $\mathcal{L}_3$ is the language of vectors $(x\|\pi_1\|\pi_2)$ such that $V_{\mathcal{L}_2}(x\|\pi_1\|\pi_2; r_1, r_2) = 1$.	$\xrightarrow{\pi_3}$	
... continue for $\rho = O(\log M)$ rounds ...		
ρa. Generate linear PCP proof π_ρ that $x \in \mathcal{L}_\rho$.	$\xrightarrow{\pi_\rho}$	ρb. Check π_ρ by explicitly running the verifier $V_{\mathcal{L}_\rho}(x\|\pi_1\|\pi_2\|\cdots\|\pi_\rho; r_1, \ldots, r_\rho)$. Verifier $V_{\mathcal{L}_\rho}$ makes $L + 2$ queries.

Figure 2.2: A depiction of the protocol flow for the linear IOP of Theorem 2.4.1. The initial language $\mathcal{L}_1$ is the language of vectors $x \in \mathbb{F}^n$ such that $\sum_{i=1}^{M} C(A_i(x)) = A(x)$, where C and $A, A_1, \ldots, A_M$ are as in Theorem 2.4.1. The expression $V_{\mathcal{L}}(x\|\pi; r)$ denotes the output of the linear PCP verifier of Theorem 2.3.3 for some language $\mathcal{L}$ run on input $(x\|\pi)$ using randomness r.

For all $i \in \{1, \ldots, \rho - 1\}$: (1) the proof π_i has length $2 \deg C + 1$, and (2) the verifier $V_{\mathcal{L}_i}$ is a circuit containing $(M/2^i)|C|$ multiplication gates. In the last step, the proof π_ρ has length $L + \deg C + 1$.

- The FLPCP verifier of Theorem 2.3.3 makes $ML/2 + 2$ linear queries and receives answers $a, a_1, \ldots, a_{ML/2}, b \in \mathbb{F}$. The verifier must then check that: (1) $a = G(a_1, \ldots, a_{ML/2})$ and (2) $b = 0$.

  The FLIOP verifier we construct does not operate exactly as the FLPCP verifier does. Instead, the FLIOP verifier we construct uses a single linear query to check that the second relation holds (i.e., that $b = 0$).

  The FLIOP verifier now outsources the work of performing the first check to the prover. More precisely:

  1. The FLIOP verifier sends to the prover the randomness r it used to generate its linear queries. In the construction of Theorem 2.3.3, this randomness consists of a single field element.
  2. This randomness defines an affine map $Q_r : \mathbb{F}^{n+m} \to \mathbb{F}^{ML/2+1}$ from the proof-input pair $(x\|\pi) \in \mathbb{F}^n \times \mathbb{F}^m$ to the query answers $(a, \bar{a}_1, \ldots, \bar{a}_{M/2}) \in \mathbb{F} \times (\mathbb{F}^L)^{M/2}$.
  3. The verifier now needs to check that $a = G(\bar{a}_1, \ldots, \bar{a}_{M/2})$. By construction of the G-gate, the verifier must check that the values $(a, \bar{a}_1, \ldots, \bar{a}_{M/2})$ satisfy a relation of the form $\sum_{i \in [M/2]} C(\bar{a}_i) = a$, But, since Q_r is an affine function of the vector $(x\|\pi)$, this is a relation of the form $\sum_{i \in [M/2]} C(A'_i(x\|\pi)) = A'(x\|\pi)$, for some affine functions $A' : \mathbb{F}^{n+m} \to \mathbb{F}$ and $A'_1, \ldots, A'_{ML/2} : \mathbb{F}^{n+m} \to \mathbb{F}^L$.

  By the induction hypothesis, the prover and verifier then can recursively invoke the FLIOP protocol to check this relation.

Security arguments. Completeness follows by construction. Soundness follows from an inductive argument. All that remains is to show that the protocol satisfies strong HVZK. To show this, note that at every level of the induction except the base case, the honest verifier makes a single query whose response should be zero. These answers are trivial to simulate. In the base case, we can invoke the strong HVZK simulator of the linear PCP of Theorem 2.3.3. □

2.4.2 A fully linear IOP for SIMD circuits

Many interesting languages are recognized by arithmetic circuits that (1) apply the same small subcircuit to each block of inputs and (2) accept iff all of the subcircuits accept. For example, the language $\mathcal{L} = \{0, 1\}^n \subseteq \mathbb{F}^n$ of zero/one vectors in $\mathbb{F}^n$ is recognized by a circuit

that on input $(x_1, \ldots, x_n) \in \mathbb{F}^n$ computes $\forall i \ b_i \leftarrow x_i(x_i - 1)$ and accepts iff all b_i are zero. We call this type of circuit a "same-instruction multiple-data" (SIMD) circuit.

Implementing the logical-AND of the n values $(b_1, \ldots, b_n)$ in an arithmetic circuit is relatively expensive. The standard technique for this requires applying the mapping $b_i \mapsto b_i^{|\mathbb{F}|-1}$ which, by Fermat's Little Theorem, takes $0 \mapsto 0$ and any non-zero value $x \mapsto 1$. Then we can take the sum $\sum_{i \in [n]} b_i$ (as long as $n < |\mathbb{F}|$) to check whether all b_i are zero. Directly applying the linear PCPs of Section 2.3 to this circuit will give a proof of size $\Omega(n \cdot \log |\mathbb{F}|)$. By adding interaction, we can shrink the proof length to $O(\log n)$ for this language at the cost of increasing the number of communication rounds to $O(\log n)$. More generally, for a SIMD circuit composed of M copies of a circuit $C : \mathbb{F}^L \to \mathbb{F}$ of degree $\deg C$, we get a proof of size $L + O(\deg C \cdot \log M)$ that has query complexity $L + O(\log M)$ and a verifier containing $O(|C|)$ multiplication gates.

Other trade-offs between proof size and verifier complexity are possible, though when compiling fully linear PCPs into proofs on secret-shared data (as in Chapter 3), we minimize the communication cost by minimizing the sum of the proof size and verifier complexity.

We give a general transformation that turns any linear IOP for a language recognized by parallel-sum circuits (e.g., Corollary 2.3.8 and Theorem 2.4.1) into a linear IOP for SIMD circuits at the cost of increasing the number of rounds by one. The idea of the construction is to replace the logical-AND operation in the SIMD circuit by a random linear combination chosen by the verifier. If any one of the SIMD subcircuits outputs a non-zero value, then with high probability the linear combination will be non-zero and the overall circuit will reject. To make the result more general, we consider circuits that apply an arbitrary linear transformation to the input wires before applying the SIMD gates to the inputs.

Definition 2.4.2 (SIMD Circuit)**.** Let $C : \mathbb{F}^L \to \mathbb{F}$ be an arithmetic circuit. Let $A_1, \ldots, A_M : \mathbb{F}^n \to \mathbb{F}^L$ be affine functions. Then define the *SIMD circuit* $C^{\mathsf{simd}}_{A_1,\ldots,A_M} : \mathbb{F}^n \to \mathbb{F}$ as: $C^{\mathsf{simd}}_{A_1,\ldots,A_M}(x) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{M} \big(C(A_i(x)) = 0\big)$ for $x \in \mathbb{F}^n$. Here, the logical-AND operator returns $0 \in \mathbb{F}$ if the statement is true, and it returns an arbitrary non-zero value otherwise.

Theorem 2.4.3 (FLIOPs for SIMD circuits)**.** *Let* Π *be a fully linear strong HVZK IOP for the language*

$$\mathcal{L}_{\widetilde{C},\tilde{A}_1,\ldots,\tilde{A}_M} = \big\{x \in \mathbb{F}^{n+M} \mid \sum_{i \in [M]} \widetilde{C}(\tilde{A}_i(x)) = 0\big\},$$

where $\widetilde{C} : \mathbb{F}^{L+1} \to \mathbb{F}$ *is an arithmetic circuit and* $\tilde{A}_1, \ldots, \tilde{A}_M : \mathbb{F}^{n+M} \to \mathbb{F}^{L+1}$ *are affine functions. Furthermore, say that* Π *has soundness error* ϵ_Π *and round complexity* ρ_Π*.*

Then, there exists a strong HVZK fully linear IOP for the language

$$\mathcal{L}_{C^{\mathsf{simd}}_{A_1,\ldots,A_M}} = \left\{x \in \mathbb{F}^n \mid C^{\mathsf{simd}}_{A_1,\ldots,A_M}(x) = 0\right\},$$

where $C : \mathbb{F}^L \to \mathbb{F}$ *is an arithmetic circuit,* $A_1, \ldots, A_M : \mathbb{F}^n \to \mathbb{F}^L$ *are affine functions, and* $C^{\mathsf{simd}}_{A_1,\ldots,A_M}$ *is as in Definition 2.4.2. The resulting linear IOP has the same proof length, query complexity, and verification circuit size as* Π*, soundness error* $\epsilon = \epsilon_\Pi + M/|\mathbb{F}|$*, and round complexity* $\rho_\Pi + 1$*.*

Proof of Theorem 2.4.3. We construct the linear IOP implied by the theorem and then prove that it satisfies the desired properties.

The idea of the proof is that in the first round of interaction, the verifier chooses a random value $r \xleftarrow{R} \mathbb{F}$. Then, in the second round, the prover uses the fully linear IOP Π to convince the verifier that the linear combination $\sum_{i\in[M]} r^i \cdot C(A_i(x))$ of the outputs of the C subcircuits is zero. If $x \notin \mathcal{L}_{C^{\mathsf{simd}}_{A_1,\ldots,A_M}}$, then at least one of the $C(\cdot)$ subcircuits outputs a non-zero value. In this case, the value of this linear combination is equal to the evaluation of a non-zero polynomial of degree at most M at the point r. Such a polynomial has at most $M/|\mathbb{F}|$ zeros, so the verifier will catch a cheating prover with good probability, over the verifier's random choice of r.

The protocol operates as follows:

- The prover's first message is the empty string $\perp$.
- The verifier's first message is a random field element $r \xleftarrow{R} \mathbb{F}$. The prover and verifier both compute the vector $(r, r^2, r^3, \ldots, r^M) \in \mathbb{F}^M$.
- Define an arithmetic circuit $C^{\mathsf{rand}}_{A_1,\ldots,A_M} : \mathbb{F}^{n+M} \to \mathbb{F}$ as:

$$C^{\mathsf{rand}}_{A_1,\ldots,A_M}(x,\ r_1, r_2, \ldots, r_M) \stackrel{\text{def}}{=} \sum_{i\in[M]} r_i \cdot C(A_i(x)) \in \mathbb{F} \qquad \text{where} \qquad \begin{array}{rl} x & \in \mathbb{F}^n \\ r_1, \ldots, r_M & \in \mathbb{F} \end{array}.$$

Now, define a circuit $\widetilde{C} : \mathbb{F}^{L+1} \to \mathbb{F}$ as $\widetilde{C}(x, r) \stackrel{\text{def}}{=} r \cdot C(x)$. Define affine functions $\tilde{A}_1, \ldots, \tilde{A}_M : \mathbb{F}^{n+M} \to \mathbb{F}^{L+1}$ such that for all $i \in \{1, \ldots, M\}$, $\tilde{A}_i(x \| r_1, \ldots, r_M) \stackrel{\text{def}}{=} (A_i(x) \| r_i)$. Then we can rewrite $C^{\mathsf{rand}}_{A_1,\ldots,A_M}$ as:

$$C^{\mathsf{rand}}_{A_1,\ldots,A_M}(x,\ r_1, r_2, \ldots, r_M) = \sum_{i \in [M]} \widetilde{C}(\tilde{A}_i(x, r_1, \ldots, r_M)) \in \mathbb{F}.$$

- Now, the prover and verifier engage in the fully linear IOP protocol Π, implied by the hypothesis of the theorem, to check that $(x \| r_1, \ldots, r_M) \in \mathbb{F}^{n+M}$ is in the language accepted by the circuit $C^{\mathsf{rand}}_{A_1,\ldots,A_M}$.

The claimed efficiency properties, along with completeness and strong HVZK, follow immediately from the construction.

To show soundness: If $x \in \mathbb{F}^n$ is not an accepting input, then the output of at least one SIMD subcircuit C is non-zero. The output of the verification circuit $C^{\mathsf{rand}}_{A_1,\ldots,A_M}$ is the value $\sum_{i \in [M]} r^i \cdot C(A_i(x))$, which is then the evaluation of a non-zero polynomial of degree at most M at a random point $r \xleftarrow{\text{R}} \mathbb{F}$. The probability then, over the verifier's choice of r, that the circuit $C^{\mathsf{rand}}_{A_1,\ldots,A_M}$ outputs zero is at most $M/|\mathbb{F}|$. Conditioned on the output of $C^{\mathsf{rand}}_{A_1,\ldots,A_M}$ being non-zero, the probability that the linear PCP verifier accepts is at most ϵ_Π. Therefore, the overall soundness error is $\epsilon = M/|\mathbb{F}| + \epsilon_\Pi$. □

Remark 2.4.4. The protocol offers a possible tradeoff between the number of the verifier's random bits and the soundness error. Instead of sending a random r the verifier can send a sequence of random elements $r_1, \ldots, r_m$ and then C_r is defined by $C_r(x) = \sum_{i \in [m]} r_i \cdot C_i(x)$. In this case the number of random bits increases by a factor of m while the additive error term falls to $1/|\mathbb{F}|$.

Applying Theorem 2.4.3 to our earlier constructions of fully linear proof systems for parallel-sum circuits immediately yields efficient proof systems for SIMD relations:

Corollary 2.4.5. *Let $C^{\mathsf{simd}}_{A_1,\ldots,A_M}$ be the SIMD circuit of Definition 2.4.2. There is a 1.5-round fully linear IOP with strong HVZK for the language $\mathcal{L}_{C^{\mathsf{simd}}_{A_1,\ldots,A_M}}$ whose efficiency parameters match those of Corollary 2.3.8.*

Corollary 2.4.6. *Let $C^{\mathsf{simd}}_{A_1,\ldots,A_M}$ be the SIMD circuit of Definition 2.4.2. There is an $O(\log M)$-round fully linear IOP with strong HVZK for the language $\mathcal{L}_{C^{\mathsf{simd}}_{A_1,\ldots,A_M}}$ whose efficiency parameters match those of Theorem 2.4.1.*

We give an example for how to use Corollary 2.4.6 to construct a strong HVZK fully linear IOP for an circuit that is useful for our applications.

Example 2.4.7. Corollary 2.4.6 gives an $O(\log n)$-round protocol for proving that a vector $x \in \mathbb{F}^n$ consists entirely of zero/one values (i.e., that $x \in \{0,1\}^n \subset \mathbb{F}^n$), with proof complexity $O(\log n)$ and a verification circuit of degree-two. In contrast, applying Theorem 2.3.3 or standard linear PCPs directly yields proofs of size $\Omega(n)$. Hence, using $O(\log n)$-rounds instead of a single round, we obtain an exponential reduction in the proof size.

Remark 2.4.8. Application of a sum-check-style protocol [203] can achieve the same complexity as the protocol of Example 2.4.7. (We thank Justin Thaler for this observation.) To sketch the protocol: the verifier first sends the prover a random vector $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$. The prover then uses a sum-check protocol to convince the verifier that $\sum_{i \in [n]} r_i x_i (x_i - 1) = 0 \in \mathbb{F}$. The asymptotic complexity of this protocol matches our own, though this protocol does not provide strong HVZK, whereas ours does. That said, it is possible to use recently developed zero-knowledge variants of the sum-check protocol to make this alternative construction zero knowledge as well [83, 276]. In future work, we hope to tease out the common structure underlying the sumcheck-based constructions and our own.

2.4.3 A fully linear IOP for low-degree languages

We can generalize the fully linear IOP for SIMD circuits to construct a fully linear IOP for any "low-degree language." These are languages that are recognized by a system of low-degree equations. The following theorem describes fully linear IOPs for such low-degree languages. Unlike the prior constructions, here we place *no restriction* on the field size, so these constructions apply even when $\mathbb{F} = \mathbb{F}_2$. Although we do not discuss it here, we have also applied this fully linear IOP to construct communication-efficient information-theoretic multiparty computation protocols in the honest-majority setting with security against malicious players.

Theorem 2.4.9 (ZK-FLIOP for low-degree languages)**.** *Let $\mathbb{F}$ be a finite field, let $C : \mathbb{F}^n \to \mathbb{F}^m$ be an arithmetic multi-output circuit of degree d defined by $C(x) = (C_1(x), \ldots, C_m(x))$ and let M be the number of distinct monomials in the representation of $C_1, \ldots, C_m$ as polynomials. Let $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid C(x) = 0^m\}$ and let ϵ be a required soundness error bound. Then, there is a fully linear IOP Π over $\mathbb{F}$ with strong HVZK for the language $\mathcal{L}_C$ that has the following efficiency features.*

- **Degree $d = 2$, constant rounds:** *If $d = 2$ then Π has 1.5 rounds, proof length $O(\eta\sqrt{n})$, and query complexity $O(\sqrt{n})$, where $\eta = \log_{|\mathbb{F}|}((m + \sqrt{n})/\epsilon)$.*
- **Degree d, logarithmic rounds:** *If $d \geq 2$ then Π has $O(\log M)$ rounds, proof length $O(\eta d \log M)$, and query complexity $O(\log M)$, where $\eta = \log_{|\mathbb{F}|}((m + d \log M)/\epsilon)$.*

Proof. Let $\mathbb{E}$ be an extension field of degree $\eta + O(1)$ over $\mathbb{F}$, write $C = (C_1, \ldots, C_m)$ over $\mathbb{E}$. Then $\mathcal{L}_C$ can be represented as a SIMD circuit, in the sense of Definition 2.4.2 with m gates over $\mathbb{E}$.

We can apply the fully linear IOP over $\mathbb{E}$ for $\mathcal{L}_C$ based on applying the SIMD transformation of Theorem 2.4.3 to one of two different proof systems, depending on the case of the theorem:

- **Degree $d = 2$, constant rounds:** We apply the SIMD transformation of Theorem 2.4.3 to the fully linear PCP of Corollary 2.3.7. The efficiency parameters, including the proof length, the query complexity and the algebraic degree of the verifier all match those of the corollary.
- **Degree d, logarithmic rounds:** We apply the SIMD transformation of Theorem 2.4.3 to the fully linear IOP of Theorem 2.4.1. Since the number of executions of C' in C_r is $O(M)$ and the degree of C' is d, the complexity measures follow and the soundness error is $O(\frac{m+\log M}{|\mathbb{F}|^{\eta+O(1)}}) \leq \epsilon$.

□

2.4.4 Fully linear IOPs inspired by the literature

We now demonstrate that a number of existing proof systems in the literature implicitly construct linear IOPs.

Short linear IOPs for low-depth circuits from Muggles. The "Muggles" protocol of Goldwasser, Kalai, and Rothblum [149, 148] is an interactive proof system for languages recognized by low-depth boolean circuits with a certain regular structure (specifically: log-space uniform circuits of polylogarithmic depth). Unlike many classical interactive proofs [16, 18, 151, 203, 248], the GKR protocol is doubly efficient: the prover runs in polynomial time and the verifier runs in *near-linear* time $n \cdot \text{polylog}(n)$, on inputs of length n.

We observe that the GKR protocol gives rise to a linear IOP for arithmetic circuit satisfiability. The benefit of the GKR-based linear IOP is that it has very low proof length,

as long as the relation in question is recognized by a shallow arithmetic circuit. Unlike in GKR's original setting of delegation of computation, here we need not impose any uniformity restrictions on the circuit.

The construction applies to *layered* arithmetic circuits of fan-in two. Consult the GKR paper [149] for a formal definition of such circuits.

We have:

Theorem 2.4.10 (Goldwasser, Kalai, and Rothblum [149, 148])**.** *Let $C : \mathbb{F}^n \to \mathbb{F}$ be a layered arithmetic circuit of fan-in two over a finite field $\mathbb{F}$ of size $|C|$ and depth d (where the size and depth also include the addition gates) and let $\mathcal{L}_C = \{x \in \mathbb{F}^n \mid C(x) = 0\}$. There exists a $O(d \cdot \log |C|)$-round fully linear IOP for $\mathcal{L}_C$ over $\mathbb{F}$ with*

- *proof length $O(d \cdot \log |C|)$ elements of $\mathbb{F}$,*
- *query complexity $O(d)$,*
- *soundness error $O(d \cdot \log |C|)/|\mathbb{F}|$, and*
- *a verifier of algebraic degree 2.*

The GKR-style linear IOP as described does *not* provide honest-verifier zero knowledge. However, using a modification proposed by Xie et al. [276, Section 4.2], it is possible to add HVZK without changing the asymptotic complexity of the fully linear IOP.

Proof idea. We sketch how we can interpret the "bare-bones" GKR protocol, as described in Rothblum's dissertation [237, Section 3.3.2], as a fully linear IOP. (Using the notation of Rothblum, we apply the GKR protocol using the parameters $|\mathbb{H}| = 2$ and $m = O(\log n)$.)

The GKR protocol runs in d phases—one for each layer of the circuit. In the each phase:

- the prover and verifier engage in a sum-check protocol on a degree-two polynomial, with a sum with $O(|C|)$ terms,
- the verifier evaluates a prover-provided linear polynomial in at two points and checks that these are equal to two values known to the verifier, and
- the verifier sends the prover a random challenge.

Each sum-check protocol requires $O(\log |C|)$ rounds of interaction, so the total number of rounds of interaction is $O(d \cdot \log |C|)$. In the fully linear IOP setting, the verifier can execute each of these sumcheck interactions using only a single linear query to the input and proofs, along with $O(1)$ additional queries at each step of the GKR protocol. Furthermore, each of these steps only requires the verifier to compute degree-two relations on the prover-provided

values. In the final phase of the the GKR protocol, the verifier must evaluate the multilinear extension of the input x at a random point, which the verifier can achieve with a single linear query. □

More recent refinements to the GKR protocol, such as those of Cormode, Mitzenmacher, and Thaler [90], Thaler [256], and Wahby et al. [264] also yield fully linear IOPs with corresponding efficiency benefits. Finally, it is possible to cast the RRR protocol [234], which gives a constant-round interactive proofs for space-bounded computations, into the fully linear IOP framework.

Constant-size linear IOPs for vectors of small Hamming weight. In certain cryptographic applications of fully linear proof systems (as in the Riposte system of Chapter 5), a prover wants to convince a verifier that a vector $x \in \mathbb{F}^n$, for a finite field $\mathbb{F}$ has low Hamming weight. In particular, define

$$\mathcal{L}_{\mathsf{HW}(d)} \stackrel{\text{def}}{=} \big\{x \in \mathbb{F}^n \mid x \text{ has at most } d \text{ non-zero entries}\big\}.$$

We give two efficient linear IOPs for $\mathcal{L}_{\mathsf{HW}(d)}$ on ideas from the literature. The first of these proof systems applies to $\mathcal{L}_{\mathsf{HW}(d)}$, for any constant weight d, but it does *not* provide zero knowledge. The second of these proof systems does provide zero knowledge, but only applies to the language $\mathcal{L}_{\mathsf{HW}(1)}$ of vectors of Hamming weight one.

In recent work [102], Damgård et al. give a Σ-protocol for proving that a vector of commitments commits to a vector with low Hamming weight. Their protocol has applications to symmetrically private information retrieval [137] and maliciously secure d-out-of-n oblivious transfer [217]. We can view their construction as a two-round linear IOP for the language of vectors in $\mathbb{F}^n$ of low Hamming weight:

Theorem 2.4.11 (Damgård, Luo, Oechsner, Scholl, and Simkin [102])**.** *Then there exists a two-round fully linear IOP for $\mathcal{L}_{\mathsf{HW}(d)}$ over a finite field $\mathbb{F}$, of characteristic greater than two, with*

- *proof length $O(d)$ elements of $\mathbb{F}$,*
- *query complexity $O(d)$,*
- *soundness error $O(n)/|\mathbb{F}|$, and*
- *a verifier of algebraic degree 2.*

The proof system does not *provide strong HVZK.*

The proof of Theorem 2.4.11 follows immediately from the construction of Damgård et al. It is possible to make the protocol of Theorem 2.4.11 strongly zero knowledge by adding an extra round of interaction. The basic idea is that the prover and verifier first engage in the protocol of Theorem 2.4.11. Before making any queries to the proof oracle, the verifier sends to the prover the random coins that the verifier *would have used* to check the proof. The prover then convinces the verifier, using a fully linear PCP with strong zero knowledge, as in Theorem 2.3.3, that the verifier would have accepted the first proof.

This next fully linear IOP recognizes the language of vectors in $\mathbb{F}^n$ of Hamming weight one, using a sketching idea of Boyle, Gilboa, and Ishai [63]. Converting their sketching scheme into a fully linear IOP is not immediate, so we describe the construction and prove its security here.

Theorem 2.4.12. *Then there exists a two-round fully linear IOP for $\mathcal{L}_{\mathsf{HW}(1)}$ over a finite field $\mathbb{F}$ that provides strong HVZK and that has*

- *proof length at most* 8 *elements of* $\mathbb{F}$*,*
- *query complexity* 4*,*
- *soundness error* $6/(|\mathbb{F}| - 2)$*, and*
- *a verifier of algebraic degree* 2*.*

Proof. We first describe the protocol and then argue for its security. The prover and verifier interact as follows:

- Let $x \in \mathbb{F}^n$ be the prover's input and let $z \in \mathbb{F}$ be the value of its single non-zero entry. The prover sends the verifier the value $z \in \mathbb{F}$ as its first proof string.
- The verifier chooses a vector of values $r = (r_1, \ldots, r_n) \in \mathbb{F}^n$ independently and uniformly at random from $\mathbb{F}^n$. The verifier sends this vector r to the prover.
- Given the verifier's challenge $r \in \mathbb{F}^n$, the prover and verifier define the following circuit

$$C_r(x, z) \stackrel{\text{def}}{=} (\langle x, r\rangle)^2 - z \cdot \langle x, R\rangle \quad \in \mathbb{F} \qquad \text{where} \qquad \begin{aligned} r &= (r_1, \ldots, r_n) \in \mathbb{F}^n \\ R &= (r_1^2, \ldots, r_n^2) \in \mathbb{F}^n. \end{aligned}$$

  In the definition of C_r, the value $\langle x, r\rangle \in \mathbb{F}$ is the inner product of the vectors $x, r \in \mathbb{F}^n$. The prover constructs a strong-HVZK fully linear PCP π asserting that the pair (x, z) is in the language recognized by the circuit C_r using, the construction of Theorem 2.3.3.

The prover sends π to the verifier.

- The verifier checks the proof π by making linear queries to the input-proof vector $(x\|z\|\pi)$.

The efficiency properties follow immediately from the construction. In particular, the length of the proof π that the prover sends in the second round has length 7, since the circuit C_r only contains two multiplication gates. Concretely, we instantiate Theorem 2.3.3 with $L = M = \deg G = 2$, so the proof length is $|z| + |\pi| = 1 + 7 = 8$ and the query complexity is 4.

We now argue that this proof system satisfies the necessary security properties.

- *Completeness.* Say that the prover's input x is the vector of all zeros in $\mathbb{F}^n$ with value $z \in \mathbb{F}$ at index $i \in [n]$. Fix a choice $r \in \mathbb{F}^n$ of the verifier's random challenge. Then we will have $C_r(x, z) = (zr_i)^2 - (zr_i^2) \cdot z = 0 \in \mathbb{F}$. Then (x, w) is in the language that C_r recognizes and by the completeness of the underlying linear PCP for C_r, the verifier will always accept.
- *Soundness.* There are two possible sources of soundness error: (1) the vector $x \notin \mathcal{L}_{\mathsf{HW}(1)}$ but (x, z) is in the language recognized by the circuit C_r and (2) the pair (x, z) is not in the language recognized by the circuit C_r and yet the verifier accepts the proof π. We bound both of these probabilities from above.

  To bound the probability of the first bad event. Fix a vector $x \in \mathbb{F}^n$ that has Hamming weight greater than one and fix any value $z \in \mathbb{F}$. Then, consider the n-variate polynomial f:

$$f(r_1, \ldots, r_n) \stackrel{\text{def}}{=} \left(\sum_{i \in [n]} x_i r_i\right)^2 - z \cdot \left(\sum_{i \in [n]} x_i r_i\right) \qquad \in \mathbb{F}[r_1, \ldots, r_n].$$

  If x has Hamming weight more than one, this polynomial cannot be identically zero, no matter the choice of $z \in \mathbb{F}$. To see this, observe that if the vector has weight-w, then the left-hand squared sum has w^2 distinct non-zero terms and the right-hand sum can have at most w distinct non-zero terms. (The coefficients of the left-hand sum have the form $2x_i x_j$; when $\mathbb{F}$ has characteristic greater than two, these coefficients are non-zero.) For any $w > 1$, f will then not be identially zero.

  Then, for a random choice of $r = (r_1, \ldots, r_n) \xleftarrow{\text{R}} \mathbb{F}^n$, the Schwartz-Zippel Lemma [242, 283] asserts that $\Pr[f(r) = 0] \leq 2/|\mathbb{F}|$, over the random choice $r \xleftarrow{\text{R}} \mathbb{F}^n$, since f has

total degree two.

The probability of the second bad event—that the verifier accepts a false proof π in the second step—is bounded by $4/(|\mathbb{F}| - 2)$, by the soundness of the underlying fully linear PCP of Theorem 2.3.3. Here, we instantiate Theorem 2.3.3 with $M = \deg G = 2$.

The total soundness error is then at most $2/|\mathbb{F}| + 4/(|\mathbb{F}| - 2) \leq 6/(|\mathbb{F}| - 2)$.

- *Strong HVZK.* We must construct a simulator S that takes no inputs and that samples from the distribution of the linear IOP verifier's view in the interaction. The simulator operates as follows:

  1. Choose a random string $r \xleftarrow{R} \mathbb{F}^n$.
  2. Run $\mathsf{view} \leftarrow S_{\mathsf{FLPCP}}()$, where S_{FLPCP} is the simulator implied by the strong HVZK property of the fully linear PCP used in the second step.
  3. Output (r, view).

  We must argue that the simulator S samples from the correct distribution. First, the simulator samples the verifier's random challenge $r \in \mathbb{F}^n$ as in the real protocol. Next, we observe that for the honest prover's first message $z \in \mathbb{F}$, and for every choice of the verifier's random challenge $r \in \mathbb{F}^n$, the vector (x, z) is in the language that the circuit C_r recognizes. In this case, the guarantee of the underlying simulator S_{FLPCP} is that it outputs samples from the honest linear PCP verifier's view of its interaction with the proof oracle. Thus, the joint distribution (r, view) is exactly as in the real interaction.

□

Remark 2.4.13 (Gracefully handling longer elements). In some applications, such as Riposte (Chapter 5), we are interested in proving that a dimension-n vector of ℓ-tuples of field elements (i.e., a vector in $(\mathbb{F}^\ell)^n$) contains a single non-zero tuple. We can directly apply the protocol of Theorem 2.4.12 to prove such statements by viewing each tuple an as element of an extension $\mathbb{E} = \mathbb{F}^\ell$. This is relatively is inefficient, since the prover and verifier must perform many operations in the large field $\mathbb{E}$.

A better approach is to add one more round of interaction at the start of the protocol, in which the verifier sends the prover a random linear map $h : \mathbb{F}^\ell \to \mathbb{F}$. Both parties then apply this map to each element of the input vector $x = (x_1, \ldots, x_n) \in (\mathbb{F}^\ell)^n$ to get a vector $x' = (h(x_1), \ldots, h(x_n)) \in \mathbb{F}^n$. With high probability over the verifier's random choice of h, the vector $x' \in \mathbb{F}^n$ has Hamming weight one if and only if the original vector x had Hamming

weight one.

The prover and verifier then execute the protocol of Theorem 2.4.12 on the smaller input $x' \in \mathbb{F}^n$. The soundness error increases by an additive $n/|\mathbb{F}|$ term to account for the chance that $x_i \neq 0$ but $h(x_i) = 0$. Apart from the cost of hashing x down to x', the complexity of the protocol is independent of the length ℓ.

2.5 Proving optimality using communication complexity

We now show a connection between fully linear PCPs and Merlin-Arthur communuication complexity [1, 185] that allows us to prove lower bounds on the size and query complexity of fully linear PCPs. One advantage of restricting our focus to fully linear PCPs (rather than arbitrary linear PCPs), is that we can prove unconditional lower bounds on their efficiency. In contrast, proving lower bounds on the size of linear PCPs necessarily implies some complexity assumption, since if P = NP, all languages in NP have trivial linear PCPs (i.e., the verifier can check whether $x \in \mathcal{L}$ without needing to access a proof).

In this section, we prove the following result, which puts a lower bound on the efficiency of fully linear PCPs:

Theorem 2.5.1. *Let $\mathcal{L}_{\mathsf{IP}} \subseteq \mathbb{F}^n \times \mathbb{F}^n$ be the language of pairs of vectors $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$ such that $\sum_{i \in [n]} x_i y_i = 0 \in \mathbb{F}$. Any fully linear PCP for $\mathcal{L}_{\mathsf{IP}}$ over a finite field $\mathbb{F}$ of size greater than n and soundness error at most $1/3$ must have proof size $|\pi|$ and query complexity ℓ such that $|\pi| + \ell = \Omega(\sqrt{n})/\log|\mathbb{F}|$.*

Remark 2.5.2. The technique we use to prove Theorem 2.5.1 is very similar to the technique that Chakrabarti et al. [76] use to get lower bounds on the efficiency of stream annotation systems (see discussion in Section 2.1). In fact, we could also prove Theorem 2.1 by showing that any fully linear PCP for $\mathcal{L}_{\mathsf{IP}}$ implies a stream annotation system for $\mathcal{L}_{\mathsf{IP}}$. To make this discussion self contained, we prove the theorem directly.

To prove Theorem 2.5.1, we show a relationship between fully linear PCPs and Merlin-Arthur communication games [1, 185]. In the Merlin-Arthur communication complexity game, Alice holds an input $x \in \{0,1\}^n$, Bob holds an input $y \in \{0,1\}^n$, and Merlin holds both x and y. Merlin sends a proof π to Alice and Bob asserting that $f(x, y) = 1$, for some $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. Alice and Bob then communicate to check the proof (using shared randomness, perhaps). If $f(x, y) = 1$, there should exist a proof that causes Alice and

Bob to accept with probability at least 2/3, over the coins of the players. If $f(x, y) = 0$, then Alice and Bob must reject all proofs with probability at least 2/3.

The *MA communication complexity* of f is the minimum number of bits that the parties must communicate to implement such a protocol (See Klauck [185] for a formal definition.) The *randomized communication complexity* of f is the minimum number of bits that the parties must communicate to implement such a protocol with no proof from Merlin.

We then have:

Lemma 2.5.3. *If there is a fully linear PCP for the language* $\mathcal{L}_f = \{(x, y) \subseteq \mathbb{F}^n \times \mathbb{F}^n \mid f(x, y) = 1\}$ *over a finite field* $\mathbb{F}$ *with proof length* $|\pi|$*, query complexity* ℓ*, and soundness error at most* $1/3$*, then the function* f *has Merlin-Arthur communication complexity at most* $(|\pi| + \ell) \log |\mathbb{F}|$ *bits.*

Proof. We construct an MA protocol for $\mathcal{L}_f$. On input $(x, y) \in \mathbb{F}^n \times \mathbb{F}^n$, Merlin runs the linear PCP prover for $\mathcal{L}_f$ to get the proof π. Merlin sends the proof to Alice and Bob using $|\pi| \log |\mathbb{F}|$ bits.

Alice and Bob use their common source of randomness to generate the linear PCP queries $q_1, \ldots, q_\ell \in \mathbb{F}^{2n+|\pi|}$. We can write the ith query q_i as a triple $(q_{i,x}, q_{i,y}, q_{i,\pi}) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^{|\pi|}$. Then the answer to the ith query is

$$a_i = \langle q_i, (x \| y \| \pi) \rangle = \langle q_{i,x}, x \rangle + \langle q_{i,y}, y \rangle + \langle q_{i,\pi}, \pi \rangle = a_{i,x} + a_{i,y} + a_{i,\pi} \quad \in \mathbb{F},$$

for $a_{i,x}, a_{i,y}, a_{i,\pi} \in \mathbb{F}$.

For each $i \in [\ell]$, Alice can compute $a_{i,\pi}$ and $a_{i,x}$ on her own. For each $i \in [\ell]$, Bob can compute $a_{i,y}$ on his own and can send these values to Alice using $\ell \cdot \log |\mathbb{F}|$ bits. Given these values, Alice can use the linear PCP decision routine to decide whether $(x, y) \in \mathcal{L}_f$. The total communication complexity is $(|\pi| + \ell) \cdot \log |\mathbb{F}|$. □

To prove Theorem 2.5.1, we need the following definition:

Definition 2.5.4 (Disjointness)**.** Define the function $f_{\mathsf{Disj}} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ to be the one that, on inputs $x_1 x_2 \ldots x_n \in \{0,1\}^n$ and $y_1 y_2 \ldots y_n \in \{0,1\}^n$, outputs 0 if and only if there exists an index $i \in [n]$ such that $x_i = y_i = 1$.

Now we can use Lemma 2.5.3 to prove that the linear PCP of Corollary 2.3.7 has optimal proof length and query complexity, up to a logarithmic factor in the field size:

Proof of Theorem 2.5.1. Towards a contradiction, assume that there exists a fully linear PCP for $\mathcal{L}_{\mathsf{IP}}$ over a finite field of size greater than n with soundness error $1/3$ and proof size and query complexity $o(\sqrt{n})/\log|\mathbb{F}|$. By Lemma 2.5.3, such a linear PCP implies the existence of an MA protocol for function $f_{IP} : \mathbb{F}^n \times \mathbb{F}^n \to \{0,1\}$ that returns "1" if and only the two input vectors have a inner product equal to zero. Furthermore, by Lemma 2.5.3, this MA protocol has communication complexity $\log|\mathbb{F}| \cdot o(\sqrt{n})/\log|\mathbb{F}| = o(\sqrt{n})$.

An MA protocol for f_{IP} over a field $\mathbb{F}$ of size greater than n suffices for deciding disjointness on n-bit strings [1]. Thus, we have constructed an MA protocol for disjointness on n-bit strings that has communication complexity $o(\sqrt{n})$. Klauck [185] has proved that any MA protocol for disjointness on n-bit strings requires communication $\Omega(\sqrt{n})$, which yields a contradiction. □

Aaronson and Wigderson [1] show that the MA communication complexity of $\mathcal{L}_{\mathsf{IP}}$ (and thus of the disjointness problem) is $O(\sqrt{n}\log n)$. Combining the linear PCP of Corollary 2.3.7 with the statement of Lemma 2.5.3 and the observation that deciding inner product is enough to decide disjointness, yields this result as a special case:

Corollary 2.5.5 (Corollary of Corollary 2.3.7 and Lemma 2.5.3)**.** *The MA communication complexity of f_{Disj} is $O(\sqrt{n}\log n)$.*

Chapter 3

Zero-knowledge proofs on distributed or secret-shared data

In this chapter, we apply the fully linear proof systems of Chapter 2 to construct a new type of zero-knowledge proof that applies to settings in which there are multiple verifiers, and each verifier holds only a *piece* of the statement. Using these proofs, a prover can convince a set of verifiers that an input $x \in \{0,1\}^*$ is in a language $\mathcal{L} \subseteq \{0,1\}^*$, even if no single verifier holds the input x in its entirety. We will give two applications of such proofs in Part II of this dissertation. The proof systems we develop in this chapter also have application to recent communication-efficient secure multi-party computation protocols [54], and we anticipate further applications in future work.

3.1 Model and background

In the classical setting for interactive zero-knowledge proofs [151], a single prover P interacts with a single verifier V. Both the prover and the verifier hold a common input x, and the prover's goal is to convince the verifier that $x \in \mathcal{L}$ for some language $\mathcal{L}$. In the typical case of an NP language $\mathcal{L}$, the proof system applies to some *NP-relation* $\mathcal{R}(x, w)$, in which only the prover P knows the witness w. In this standard setting, we say that the proof system is zero knowledge if it hides the witness w from the verifier. A primary motivation for our notion of fully-linear proof systems is to capture the setting in which the proof system also hides the *input* x from the verifier. In our setting, the notion of zero-knowledge if meaningful even for polynomial-time languages $\mathcal{L}$.

Our distributed model. We consider a distributed model in which a single prover P interacts with *multiple* verifiers $V_1, V_2, \ldots, V_s$ over a network that includes secure point-to-point channels. Each verifier V_j holds a *piece* $x^{(j)} \in \mathbb{F}^{n_j}$ of an input $x = x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)} \in \mathbb{F}^n$, and the prover's task is to convince the verifiers that the concatenated input x is in some language $\mathcal{L} \subseteq \mathbb{F}^n$. We will assume the prover knows x in its entirety. Informally, a zero-knowledge proof on distributed data is an interactive protocol between P and the s verifiers V_j that satisfies the following properties:

- The proof system must be *complete*, in the sense that if $x \in \mathcal{L}$ and all parties are honest, all verifiers accept.
- The proof system must be *sound*, in the sense that if $x \notin \mathcal{L}$, then all verifiers reject with high probability.
- Finally, the proof system must have *strong zero knowledge*: any proper subset of the verifiers should learn no additional information about x, apart from what follows from their own pieces and the fact that $x \in \mathcal{L}$.

Note that this notion of zero knowledge is stronger than the traditional one. In a standard zero-knowledge interactive proof, the verifier learns the statement x and learns that $x \in \mathcal{L}$, but the verifier learns nothing else. In our setting, *any proper subset of the verifiers does not even learn the statement* x: such a collection of verifiers just learn that they are jointly holding pieces $x^{(j)}$ of some input $x \in \mathcal{L}$.

Up to now, we have only considered the honest-verifier setting, in which we require soundness to hold against a potentially malicious prover, but (strong) zero knowledge is required only with respect to honest verifiers. For later applications, we will need to consider a more stringent adversarial setting, in which *either* the prover *or* all but one of the verifiers are malicious. That is, we will require:

- *soundness* to hold against a malicious prover when all verifiers are honest verifiers, and
- *zero knowlegde* to hold against a coalition of up to $s - 1$ adversarial verifiers.

In joint work with Boneh, Boyle, Gilboa, and Ishai [54], we extend these protocols to the setting in which soundness holds even when the prover colludes with some subset of the verifiers. In this dissertation, we will restrict ourselves to the simpler setting in which the prover does not collude with the verifiers.

We now formalize the above discussion. We start by defining distributed analogues of the basic objects of zero-knowledge proofs: inputs, languages, and relations. As before, we focus

for simplicity on the finite case, where languages are subsets of vectors in $\mathbb{F}^n$, and we ignore issues of computational efficiency. Except when explicitly noted otherwise, all definitions and results extend in a natural way to the usual asymptotic framework with polynomial-time parties and relations.

Definition 3.1.1 (Distributed inputs, languages, and relations)**.** Let s be a number of parties, let $\mathbb{F}$ be a finite field, and let $n, n_1, \ldots, n_s \in \mathbb{N}$ be length parameters, where $n = n_1 + n_2 + \ldots + n_s$. An *$s$-distributed input over* $\mathbb{F}$ (or just a distributed input) is a vector $x = x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)} \in \mathbb{F}^n$, where $x^{(i)} \in \mathbb{F}^{n_i}$. We refer to each $x^{(i)}$ as a *piece* of x. (In the context of secret sharing, we will sometimes refer to $x^{(i)}$ as a *share*.) An *s-distributed language* $\mathcal{L}$ is a set of s-distributed inputs. A *distributed NP relation* with witness length h is a binary relation $\mathcal{R}(x, w)$ where x is a s-distributed input and $w \in \mathbb{F}^h$. We assume that all x in $\mathcal{L}$ and $(x, w) \in \mathcal{R}$ share the same length parameters. Finally, we let $\mathcal{L}(\mathcal{R}) = \{\, x : \exists w \, (x, w) \in \mathcal{R} \,\}$.

We now define our protocol model for zero-knowledge proofs on distributed data. Our default network model allows synchronous communication over secure point-to-point channels. We will later also allow protocols to use an ideal broadcast primitive and an ideal coin-tossing primitive.

Definition 3.1.2 (s-verifier interactive proof protocol)**.** A *s-verifier interactive proof protocol* over $\mathbb{F}$ is an interactive protocol $\Pi = (P, V_1, \ldots, V_s)$ involving a prover P and s verifiers $V_1, V_2, \ldots, V_s$. The protocol proceeds as follows.

- In the beginning of the protocol the prover holds an s-distributed input $x = x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)} \in \mathbb{F}^n$ and possibly a witness $w \in \mathbb{F}^h$. Each verifier V_j holds an input piece $x^{(j)}$.
- The protocol allows the parties to communicate in synchronous rounds over secure point-to-point channels. The role of each party in this interaction is defined by a *next message function*, which specifies the messages it sends in each round as a function of its local input, random input, and messages received in previous rounds. (The parties pick their random inputs independently from a distribution that will be implicit in the protocol description.) While honest parties send messages according to Π, malicious parties, which a central adversary controls, can send arbitrary messages. Moreover, the messages that malicious parties send in each round can depend on all messages they receive from honest parties in the same round (namely, the adversary has a *rushing* capability).

- After some fixed number of rounds, the protocol terminates. Upon termination, each verifier outputs either "accept" or "reject" based on its *view*, where the view of a verifier V_j consists of its input piece $x^{(j)}$, random input $r^{(j)}$, and messages it received during the protocol execution. To ensure that all honest verifiers either simultaneously accept or reject, we will assume by default that the decision is determined by *public information* that was communicated over a broadcast channel (see below).

We denote by $\Pi(x, w)$ the probabilistic experiment of running Π on distributed input x and witness w, and say that $\Pi(x, w)$ *accepts* if, in the end of this experiment, all verifiers output "accept." Otherwise, we say that $\Pi(x, w)$ *rejects*. We denote by $\mathsf{View}_{\Pi,T}(x, w)$ the (joint distribution of) views of verifiers V_j with $j \in T$ in the execution of Π on the distributed input x and witness w.

Helper functionalities. For modularity, we factor out two standard primitives that are useful for obtaining security against malicious verifiers: a *broadcast* primitive, allowing a party to send an identical message to all parties (where even a malicious sender cannot make different honest parties receive different messages), and a *coin-tossing* primitive, which generates unpredictable, public randomness. We model these two primitives by using ideal multi-party functionalities formally defined below. We assume by default that a s-verifier interactive proof protocol can make oracle calls to these functionalities.

Definition 3.1.3 (Ideal broadcast and coin-tossing functionalities)**.** We define the two helper functionalities $\mathcal{F}_{\mathsf{bcast}}$ and $\mathcal{F}_{\mathsf{coin}}$ as follows:

- The *broadcast* functionality $\mathcal{F}_{\mathsf{bcast}}$ receives an input message m from a verifier V_j and delivers the output (j, m) to all parties. When considering protocols that should be secure in the presence of malicious verifiers, we assume that each verifier's decision whether to accept or reject is determined in the same way based on (public) messages sent via $\mathcal{F}_{\mathsf{bcast}}$.
- The *coin-tossing* functionality $\mathcal{F}_{\mathsf{coin}}^{\mathcal{S}}$, for a finite set $\mathcal{S}$, is a randomized, input-less functionality that outputs to all parties (prover and verifiers) a uniformly random element $r \in_R \mathcal{S}$.

The requirement that accepting or rejecting depends only on public information ensures agreement between honest verifiers on whether to accept, and effectively forces a simulator of the view of malicious verifiers to know whether honest verifiers accept.

When a protocol Π is allowed access to these helper functionalities, we say that the protocol runs in the $(\mathcal{F}_{\mathsf{bcast}}, \mathcal{F}_{\mathsf{coin}})$-*hybrid model*.

On realizing $\mathcal{F}_{\mathsf{coin}}$ and $\mathcal{F}_{\mathsf{bcast}}$. When all verifiers are honest, it is trivial to implement both functionalities. For instance, $\mathcal{F}_{\mathsf{coin}}^{\mathcal{S}}$ can be realized by having V_1 pick r uniformly from $\mathcal{S}$ and send it to all other parties. When the verifiers can be malicious, it is possible to use standard composable implementations of these primitives over secure point-to-point channels. These implementations offer different trade-offs between security type and efficiency. (See, for example, Goldwasser and Lindell [150], Cohen et al. [89], and the references therein.) Our protocols will make a minimal use of these two ideal primitives. In general, our protocols will invoke these ideal primitives in total on inputs that are sublinear in the size of the distributed input x. Thus, for distributed zero-knowledge proofs on large inputs, the implementation of these helper functionalities will not present an efficiency bottleneck.

For the purpose of measuring round complexity of protocols, we will separately count "standard" rounds of point-to-point communication and rounds in which the above oracles are invoked. For communication complexity, we will count the length of the input to $\mathcal{F}_{\mathsf{bcast}}$ or output of $\mathcal{F}_{\mathsf{coin}}^{\mathcal{S}}$.

We now present our definition of zero-knowledge proof on distributed data:

Definition 3.1.4 (Zero-knowledge proofs on distributed data: malicious prover *or* verifiers)**.** Let $\mathcal{R}(x, w)$ be an s-distributed relation over finite field $\mathbb{F}$. We say that a s-verifier interactive proof protocol $\Pi = (P, V_1, \ldots, V_s)$ is a *distributed strong zero-knowledge proof protocol for $\mathcal{R}$ with security against malicious prover* or t *verifiers* and soundness error ϵ, if Π satisfies the following:

- **Completeness.** For every s-distributed input $x = x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)} \in \mathbb{F}^n$ and witness $w \in \mathbb{F}^h$ such that $(x, w) \in \mathcal{R}$, the execution of $\Pi(x, w)$ accepts with probability 1.
- **Soundness.** For every malicious prover P^*, every s-distributed input x such that $x \notin \mathcal{L}(\mathcal{R})$, and every witness w, the execution of $\Pi^*(x, w)$ rejects, except with probability ϵ. Here, Π^* denotes the interaction of P^* with the honest verifiers $(V_1, \ldots, V_s)$.

We define a zero-knowledge property similar to the one used in our linear IOP definition:

- **Strong zero knowledge.** For every $T \subseteq [s]$ of size $|T| \leq s - 1$ and a malicious adversary A controlling the verifiers $\{V_j\}_{j \in T}$, there exists a simulator S such that for every k-distributed input $x = x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)}$ and witness w such that $(x, w) \in \mathcal{R}$

we have:

$$S\big(\{(j, x^{(j)}) \mid j \in T\}\big) \quad \equiv \quad \mathsf{View}_{\Pi^*,T}(x, w),$$

where here Π^* denotes the interaction of the malicious adversary A with the honest prover P and the honest verifiers $\{V_j\}_{j \notin T}$.

Remark 3.1.5 (Honest-verifier variant). One may consider a relaxed notion of strong *honest-verifier* zero knowledge, defined identically to the above, but where the subset of verifiers $\{V_j^*\}_{j \in T}$ is stipulated to honestly follow the protocol.

If we do not consider a zero-knowledge requirement, then any interactive proof system for $\mathcal{R}$ immediately yields an s-verifier interactive proof system for $\mathcal{R}$: Indeed, verifiers $V_2, \ldots, V_s$ can simply send their pieces of x to V_1, who then engages with P in a standard interactive proof protocol for $\mathcal{R}$. This approach does not respect strong zero knowledge. Moreover, it cannot not meet our goal of making communication complexity sublinear in the input length. We now show a similar compiler that allows us to get around both limitations by starting with a public-coin strong HVZK fully linear IOP.

3.2 Construction from fully linear proof systems

At a high level, we convert a public-coin, fully linear IOP with strong HVZK Π into a zero-knowledge proof on distributed data Π_{dist} by exploiting the fact that it is easy to distribute the task of answering public linear queries. Recall that the prover holds $(x, w) \in \mathcal{R}$, where each verifier holds a different piece $x^{(j)}$ of the input x. The prover in Π_{dist} will execute the fully linear IOP on its input (x, w), where in each round i she *splits* the proof π_i for this round into additive secret shares, and sends one share to each verifier over a secure channel. The verifiers sample the fresh public random challenge r_i, which determines the query vectors $q_{i,1}, \ldots, q_{i,\ell_i}$, using the ideal coin-tossing oracle $\mathcal{F}_{\mathsf{coin}}$. Since each query defines a linear combination of the input x and the proof π_i, the verifiers can use their input pieces and additive shares of π_i to compute additive shares of the answers to the queries, which they send to V_1. At the conclusion, V_1 reconstructs the answers a_i, feeds them into the IOP decision predicate, and broadcasts the decision ("accept" or "reject") to the other verifiers.

The above construction directly preserves completeness as well as soundness against a malicious prover since, in essence, the secret shares of each π_i commit the prover to a fixed proof. Proving zero knowledge against $s - 1$ malicious verifiers is more subtle. The random

challenges generated by $\mathcal{F}_{\mathsf{coin}}$ and the proof shares received from the prover are easy to simulate. To complete the simulation, we distinguish between two cases. If V_1 is not corrupted, then we only need to simulate the final decision that V_1 broadcast Since other verifiers can be malicious, they have control over the answers a_i that V_1 reconstructs. However, since it is possible to simulate the honest answers a_i without knowing the input (by the strong zero knowledge of Π), and since the verifiers can predict the effect of their misbehavior on the answers that V_1 reconstructs based on their inputs, it is possible to simulate the final decision that V_1 computes. For the case in which V_1 is corrupted, we can assume without loss of generality that in each round i of Π, each set of queries (viewed as linear functions of the input x together with the proof vector $\pi = \pi_1 \| \pi_2 \| \cdots$) are linearly independent. Then we use the strong zero-knowledge property to deduce that linear independence must hold even when restricted to the π_i coefficients. The latter implies that the answer shares of any strict subset of the verifiers are uniformly and independently distributed, which allows us to obtain a distributed simulator for Π_{dist} from the IOP simulator.

We now give the formal description of Π_{dist} and its analysis. We assume that $\Pi = (P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}})$ is a ρ-round public-coin fully linear IOP for $\mathcal{R}$ (viewed as a non-distributed relation) with proof length $m = \sum_{i \in [\rho]} m_i$, random challenges $r_i \in \mathbb{F}^{b_i}$, query complexity ℓ, soundness error ϵ, and strong HVZK.

Protocol Π_{dist}:

- **Inputs:** The prover P has input (x, w), where $x = x^{(1)} \| \cdots \| x^{(s)}$. Each verifier V_j, for $1 \leq j \leq s$, has input piece $x^{(j)}$.
- **Interaction:** For each round $i = 1, 2, \ldots, \rho$:
  - If $i = 1$, the prover P lets $(\pi_1, \mathsf{st}_1^P) \leftarrow P_{\mathsf{FLIOP}}((x, w), \perp)$, else it lets $(\pi_i, \mathsf{st}_i^P) \leftarrow P_{\mathsf{FLIOP}}(\mathsf{st}_{i-1}^P, r_{i-1})$.
  - The prover P splits π_i into additive shares by sampling random $\pi_{i1}, \ldots, \pi_{is} \in \mathbb{F}^{m_i}$ such that $\pi_i = \sum_{j \in [s]} \pi_{ij} \in \mathbb{F}^{m_i}$ and sends the proof share π_{ij} to V_j.
  - The $s + 1$ parties invoke the coin-tossing functionality and set $r_i \leftarrow \mathcal{F}_{\mathsf{coin}}^{\mathcal{S}}$ for $\mathcal{S} = \mathbb{F}^{b_i}$.
- **Queries:**
  - All s verifiers let $(q_{i,1}, \ldots, q_{i,\ell})$ be the query vectors determined by $r_1, \ldots, r_\rho$. Each

of these query vectors is in $\mathbb{F}^{n+m}$. Let $Q \in \mathbb{F}^{(n+m)\times\ell}$ be the matrix whose columns are the queries $(q_{i,1}, \ldots, q_{i,\ell})$, and let $Q^{(j)} \in \mathbb{F}^{(n_j+m)\times\ell}$ be the matrix Q with the first n rows restricted to the $x^{(j)}$-entries.

– Each verifier V_j sends to V_1 the answer share $a_j \leftarrow (x^{(j)} \| \pi_1 \| \cdots \| \pi_\rho) \cdot Q^{(j)}$ and V_1 computes $a \leftarrow \sum_{j\in[s]} a_j$, where $a \in \mathbb{F}^\ell$.

- **Outputs:** Verifier V_1 applies the decision predicate of V_{FLIOP} to the random challenges $r_1, \ldots, r_\rho$ and the answers $a \in \mathbb{F}^\ell$, and broadcasts its decision ("accept" or "reject"). All verifiers output the decision broadcasted by V_1.

Efficiency. Note that the communication between verifiers in the end of round i can be merged with the prover's communication to the verifiers in the beginning of round $i+1$. This gives an implementation of Π_{dist} with the following efficiency features:

- The round complexity involves $\rho + 1$ rounds of point-to-point communication, as well as ρ sequential invocations of $\mathcal{F}_{\mathsf{coin}}$;
- The communication includes a total of $ms + \ell - 1$ field elements and a single bit of broadcast;
- There are ρ invocations of $\mathcal{F}_{\mathsf{coin}}$ that generate a total of $b = \sum_{i=1}^{\rho} b_i$ random field elements;
- The arithmetic circuit complexity of P is $\sigma_P + O(ms)$, where σ_P is the arithmetic circuit complexity of P_{FLIOP}. Similarly, the arithmetic circuit complexity of V is $\sigma_V + O(\ell s)$, where σ_V is the arithmetic circuit complexity of V_{FLIOP}.

We now state and prove the security properties of Π_{dist}.

Theorem 3.2.1 (Zero-knowledge proof on distributed data: malicious prover *or* verifiers)**.** *Let $\mathcal{R}$ be an s-distributed relation over $\mathbb{F}$. Suppose $\Pi = (P_{\mathsf{FLIOP}}, V_{\mathsf{FLIOP}})$ is a ρ-round public-coin fully linear IOP for $\mathcal{R}$ (viewed as a non-distributed relation) with soundness error ϵ and strong HVZK. Then, the protocol $\Pi_{\mathsf{dist}} = (P, V_1, \ldots, V_s)$ described above is a distributed strong zero-knowledge proof protocol for $\mathcal{R}$ in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{bcast}})$-hybrid model, with security against malicious prover* or *$s-1$ verifiers and soundness error ϵ.*

Proof of Theorem 3.2.1. We separately argue completeness, soundness, and zero knowledge.

Completeness. Follows easily from the completeness of Π.

SOUNDNESS. Here we assume that the s verifiers V_j are honest but the prover P^* is malicious. For any malicious strategy of P^* in Π_{dist} wef define a corresponding malicious strategy of P^*_{FLIOP} in Π so that the probability of all verifiers rejecting in Π_{dist} is equal to the probability of V_{FLIOP} rejecting in Π. Concretely, each (badly formed) proof generated by P^*_{FLIOP} is the sum of the corresponding proof shares generated by P^* on the same random challenges. It follows that Π_{dist} has the same soundness error as Π.

STRONG ZERO KNOWLEDGE. We show that the protocol satisfies the strong zero knowledge requirement Recall that in the setting of strong zero knowledge, we consider the case in which the prover P is honest, and up to $s-1$ verifiers can be malicious. Let $T \subseteq [s]$ of size $|T| \leq s-1$ and let A be a malicious adversary controlling the verifiers $\{V_j\}_{j\in T}$. Let $x = x^{(1)}\|x^{(2)}\|\cdots\|x^{(s)}$ and w such that $(x,w) \in \mathcal{R}$. We describe a simulator S such that $S(\{(j,x^{(j)})\}_{j\in T})$ perfectly generates the view of the verifiers in T in the interaction of A with P and the honest verifiers on inputs (x,w). Note that because Π_{dist} uses the ideal coin toss functionality $\mathcal{F}_{\mathsf{coin}}$, the malicious verifiers have no effect on the choice of the random challenges. The simulation now depends on whether V_1 is corrupted.

- *Case 1:* $1 \notin T$. In this case, S starts by invoking the simulator S_{FLIOP} implied by the strong HVZK property of Π to generate the random challenges r_i for each round $1 \leq i \leq \rho$, along with the answers a. Then, for each round i and corrupted verifier V_j (with $j \in T$), the simulator S simulates the message V_j receives from P by picking a proof share π_{ij} uniformly at random from $\mathbb{F}^{m_i}$. The simulator produces the message that V_1 sends in the final round as follows. First, for each malicious verifier $j \in T$, the simulator S uses the simulated values $x^{(j)}$, $(r_1,\ldots,r_\rho)$, and $(\pi_{1j},\ldots,\pi_{\rho j})$ to compute two values: (1) the answer share a_j that verifier V_j *should* have sent to V_1 and (2) the answer a^*_j that verifier V_j *actually* sent. (The simulator obtains the latter by running A on the view containing the input pieces $x^{(j)}$ and the simulated values.) Then, S simulates the answers computed by V_1 as $a^* = a + \sum_{j\in T}(a^*_j - a_j)$, applies the decision predicate of V_{FLIOP} to the random challenges $r_1,\ldots,r_\rho$ and the answers a^*, and uses the result ("accept" or "reject") to simulate the final broadcast message received from V_1.

  To see that this perfectly simulates the view of malicious verifiers, first note that the joint simulation of $(r_1,\ldots,r_\rho)$, a, and $(\pi_{1j},\ldots,\pi_{\rho j})_{j\in T}$ perfectly emulates the real execution, since the proof shares $(\pi_{1j},\ldots,\pi_{\rho j})_{j\in T}$ in the real execution are uniformly random independently of r_i, π_i. It follows that the simulated value of $\sum_{j\in T}(a^* - a)$,

which captures the difference between the correct answer and the one obtained by V_1, is also distributed identically to the real protocol even when conditioned on all $r_i, \pi_i, \{\pi_{1j}, \ldots, \pi_{\rho j}\}_{j \in T}$. This implies the correctness of the simulation of the final decision bit of V_1.

- *Case 2:* $1 \in T$. The simulator produces the proof shares π_{ij} as before. The simulator simulates the challenges r_i and answer shares a_j jointly by using the simulator S_{FLIOP} implied by the strong HVZK property of Π. First, the simulator uses S_{FLIOP} to simulate the joint distribution of the random challenges r_i and query answers a. The challenges r_i determine a query matrix Q as in Π. Assume without loss of generality that Q has full rank. (Otherwise queries are redundant in the sense that there is a query whose answer can be inferred from the other answers.) Given the query answers a, the simulator computes the query answer shares a_j that honest verifiers V_j (with $j \notin T$) send to V_1 as follows: first, simulator computes the answer shares a_j for $j \in T$ from π_{ij} and Q as in Π_{dist}, namely $a_j \leftarrow (x^{(j)} \| \pi_{1j} \| \cdots \| \pi_{\rho j}) \cdot Q^{(j)}$. Then the simulator chooses the messages a_j, for $j \notin T$, at random subject to the restriction $\sum_{j \in [s]} a_j = a$.

  To prove that the simulation is perfect, we prove that in an honest execution of Π_{dist}, the answer shares a_j are *uniformly distributed* subject to the restriction that they sum to a. This reduces to showing that each strict subset of the a_j are uniform and independent. Somewhat unexpectedly, this relies not only on the above full rank assumption but also on the strong zero knowledge property of Π. Indeed, the description of Π_{dist} directly implies that any strict subset of the π_{ij} is uniformly distributed. Since $a_j = (x^{(j)} \| \pi_{1j} \| \cdots \| \pi_{\rho j}) \cdot Q^{(j)}$, it suffices to show that if we restrict the rows of Q to their $(\pi_{1j} \| \cdots \| \pi_{\rho j})$-entries (namely, remove the first n rows of Q), the columns are still linearly independent. Suppose towards contradiction that the columns are linearly dependent with positive probability. Conditioned on this event, we can compute (with certainty) a corresponding linear combination of the entries of x from the answers $a = (x \| \pi_1 \| \cdots \| \pi_\rho) \cdot Q$, since the π parts can be cancelled out. This contradicts the strong zero knowledge property of Π.

This concludes the proof of Theorem 3.2.1. □

To give a concrete application, consider the class of degree-d distributed languages, namely ones whose membership can be tested by a set of degree-d equations. Combining Theorem 2.4.9 with Theorem 3.2.1, we get the following corollary.

Corollary 3.2.2 (Sublinear zero-knowledge proofs for distributed low-degree languages: malicious prover *or* verifiers)**.** *Let $d \geq 2$ be a positive integer, $\epsilon > 0$ an error bound, and $\mathcal{L} \subseteq \mathbb{F}^n$ an s-distributed language whose membership is tested by m degree-d polynomials. Then, there exists an $O(d \log n)$-round strong zero-knowledge proof protocol for $\mathcal{L}$ in the $(\mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{bcast}})$-hybrid model, soundness error ϵ, and communication complexity of $O(sd^2 \log n \cdot \log(m/\epsilon))$ field elements.*

3.3 Application: Proofs on secret-shared data

Several recent works, including [63, 93], consider a setting in which a client secret-shares a large input x among two or more servers, and the servers want to verify that the client-provided input "well-formed" in some application-specific sense. (We discuss these applications further in Chapters 4 and 5.) The notion of zero-knowledge proofs on distributed data can capture this setting as a special case. Here, we let the client play the role of the prover P and each of the s servers play the role of a verifier V_i. The input pieces $x^{(1)}, \ldots, x^{(s)}$ are the shares of x that the client generated using some secret-sharing scheme and distributed amongst the servers.

We say that a secret sharing scheme is *t-private* if any set of t shares reveal nothing about the secret. Our notion of zero-knowledge proofs on distributed data can be used to prove statements on the secret-shared input x, while completely hiding x from any set of t verifiers, in the following generic way. Let $\mathcal{R}$ be an NP relation and let $\mathcal{L}(\mathcal{R})$ be the corresponding NP language. To prove that $x \in \mathcal{L}(\mathcal{R})$, for a (non-distributed) relation $\mathcal{R}(x, w)$, the prover P and the s verifiers V_i engage in a zero-knowledge proof for the s-distributed relation $\mathcal{R}'$ defined by $\mathcal{R}'(x^{(1)} \| x^{(2)} \| \cdots \| x^{(s)}, w) = \mathcal{R}(\mathsf{Rec}(x^{(1)}, \ldots, x^{(s)}), w)$, where Rec is the reconstruction algorithm of the secret-sharing scheme. If the scheme is t-private and we use a zero-knowledge proof on distributed data with security against t verifiers (such as the ones obtained via Theorem 3.2.1), a set of t malicious verifiers cannot learn any information about the secret x other than the fact that $x \in \mathcal{L}(\mathcal{R})$. Indeed, it is possible to simulate the view of the t verifiers from scratch by first simulating their shares (which we can do by sharing a dummy secret) and then invoking the distributed zero-knowledge simulator on these shares. Note that for the typical case of *linear* secret sharing scheme, the algebraic degree of $\mathcal{R}'$ is the same as that of $\mathcal{R}$. Thus, we can apply our sublinear IOPs for low-degree languages to get sublinear protocols for proving in zero-knowledge that a secret-shared input x satisfies a set

of low-degree constraints.

For the case where each entry of x is secret-shared independently using some linear secret-sharing scheme (for instance, x is randomly split into s additive shares $x^{(1)}, \ldots, x^{(s)}$ such that $x = x^{(1)} + \ldots + x^{(s)}$), it is possible to simplify the above approach as follows. Instead of defining a new relation $\mathcal{R}'$ that includes the reconstruction of x from its shares, one can directly apply a fully linear IOP for $\mathcal{R}$ to the shares of x by letting each verifier locally apply each IOP query to the concatenation of its share of x and the share of the proof. This leaner variant can be proved secure by following the outline of the security proof of the protocol Π_{dist} from Theorem 3.2.1.

3.3.1 Generically achieving honest-verifier zero knowledge

Theorem 3.2.1 compiles a fully linear IOP with strong HVZK into a strong zero-knowledge proof on distributed data. It turns out that if we settle for strong *honest-verifier* zero knowledge, we can get a similar conclusion (with a small additional overhead) even if we start with a fully linear IOP that does not provide *any* form of zero knowledge.

The high level idea, which prior work developed in related contexts [63, 93], is to avoid directly reconstructing the answers a to the fully linear PCP queries. Instead we apply a general-purpose MPC protocol that allows the verifiers to compute the answer to the FLPCP decision predicate from the answer shares a_j. The communication complexity of this protocol scales linearly with the circuit complexity of the verification predicate, which in the cases we are interested in will be sublinear. If we assume that a majority of the verifiers are honest, it is possible to construct realize this MPC protocol directly, without any help from the prover [33]. To handle an arbitrary subset of semi-honest verifiers, the prover can provide correlated randomness to the verifiers (e.g., in the form of multiplication triples [24]) to enable MPC with dishonest majority. The verifiers then use this correlated randomness to compute the output of the linear IOP verification circuit without revealing any additional information about their inputs.

Note that if the prover is malicious, it can provide malformed correlated randomness. However, for natural MPC protocols and linear IOPs, such a malicious strategy will not give the prover any advantage in violating soundness. The high level reason is that the effect of malformed randomness is equivalent to an "additive attack" on the IOP verification circuit, and such an attack cannot reduce the entropy of the output of the verification circuit in case the proof is incorrect (see [63, 93] for more details).

The amount of correlated randomness that the prover must provide is linear in the number of multiplication gates of the linear IOP verification circuit. For example, for the GKR-based linear IOP of Theorem 2.4.10, the verifier provides $O(1)$ field elements worth of correlated randomness to each verifier in each protocol round.

Finally, we note that the above transformation fails to provide zero knowledge against *malicious* verifiers even if the MPC protocol is secure against malicious parties, since by changing their input to the IOP verification circuit and observing the output, the verifiers can potentially learn additional information about x, beyond the fact that $x \in \mathcal{L}(\mathcal{R})$.

3.3.2 Fiat-Shamir transform for proofs on distributed data

A commonly used heuristic for eliminating interaction in interactive proofs is the Fiat-Shamir transform [126]. Analyzed in the random oracle model (ROM), this transform applies to *public-coin* protocols and proceeds as follows: whenever the protocol generates a random challenge, the prover generates this challenge on its own by applying a random oracle H to the concatenation of the input x and the communication transcript up to this point. It is possible to apply the Fiat-Shamir transform, or its derivatives [39], for converting PCPs and IOPs into succinct non-interactive argument systems for NP languages [39, 210].

While it is relatively straightforward to rigorously analyze the Fiat-Shamir transform for three-move public-coin protocols (two-round protocols, in our terminology), the situation becomes much more complicated and subtle for many-round protocols [39]. We outline how a Fiat-Shamir-style transform could work for proofs on secret-shared data, though we do not provide a rigorous security analysis here.

Applying the Fiat-Shamir transform in our distributed setting poses the following difficulty. While the random challenges r_i are indeed public, no single verifier holds the entire the input or communication transcript—these are distributed amongst the verifiers. To get around this difficulty, we generate each random challenge r_i based on the *joint* view of the verifiers in previous rounds. Another advantage of the resulting protocol, other than minimizing round complexity, is that it no longer needs to rely on a coin-tossing oracle $\mathcal{F}_{\mathsf{coin}}$, or an interactive protocol that emulates it.

Our distributed variant of the Fiat-Shamir transform, when applied to the distributed zero-knowledge protocol of Theorem 3.2.1, proceeds as follows. The prover derives each random challenge r_i as the exclusive-or of s random challenges r_{ij}, where r_{ij} is obtained by

hashing the view of V_j up to this point. Concretely, let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a random hash function, where λ is a security parameter. For each round $i = 1, 2, \ldots, \rho$ and verifier index $j \in [s]$, the prover P computes $r_{ij} \leftarrow H(i, j, x^{(j)}, \nu_{ij}, \pi_{ij})$, where $\nu_{ij} \in_R \{0,1\}^\lambda$ is a random blinding value chosen by the prover, and then lets $r_i = \oplus_{j \in [s]} r_{ij}$. The blinding values ν_{ij} are used to ensure that the hashes r_{ij} leak no information about π_{ij}.

The verifiers can deterministically replay this process as long as the prover sends to each V_j the blinding values $(\nu_{1j}, \ldots, \nu_{\rho j})$ it used to generate the responses. The verifiers check that all information provided by the prover is consistent with their local views (where the proof shares are used to check the correctness of the random challenges r_{ij}) and then accept or reject based on the random challenges r_i and the reconstructed answers.

Part II

Applications

In this second part of the dissertation, we describe how to apply our new zero-knowledge proofs on secret-shared data (Chapter 3) to solve two privacy problems. The first, which we discuss in Chapter 4, is the problem of privacy-preserving collection of aggregate statistics. The second, which we discuss in Chapter 5 is the problem of anonymous messaging. The systems we construct in the following chapters are made possible by our new zero-knowledge proof constructions. As we will show, application of these new techniques results in orders-of-magnitude performance improvements over the prior work.

Chapter 4

Prio: Privacy-preserving computation of aggregate statistics

4.1 Introduction

Our smartphones, cars, and wearable electronics are constantly sending telemetry data and other sensor readings back to cloud services. With these data in hand, a cloud service can compute useful *aggregate statistics* over the entire population of devices. For example, navigation-app providers collect real-time location data from their users to identify areas of traffic congestion in a city and route drivers along the least-crowded roads [174]. Fitness-tracking services collect information on their users' physical activity so that each user can see how her fitness regimen compares to the average [164]. Web-browser vendors collect lists of unusually popular homepages to detect homepage-hijacking adware [121].

Even when a cloud service is only interested in learning aggregate statistics about its user population as a whole, such services often end up collecting private data from each client and storing it for aggregation later on. These centralized caches of private user data pose severe security and privacy risks: motivated attackers may steal and disclose clients' sensitive information [183, 270], cloud services may misuse the clients' information for profit [253], and intelligence agencies may appropriate the data for targeting or mass surveillance purposes [139].

To ameliorate these threats, major technology companies, including Apple [154] and Google [121, 124], have deployed privacy-preserving systems for the collection of user data. These systems use a "randomized response" mechanism to achieve differential privacy [115,

271]. For example, a mobile phone vendor may want to learn how many of its phones have a particular uncommon but sensitive app installed (e.g., the AIDSinfo app [261]). In the simplest variant of this approach, each phone sends the vendor a bit indicating whether it has the app installed, except that the phone flips its bit with a fixed probability $p < 0.5$. By summing a large number of these noisy bits, the vendor can get a good estimate of the true number of phones that are running the sensitive app.

This technique scales very well and is robust even if some of the phones are malicious—each phone can influence the final sum by ± 1 at most. However, randomized-response-based systems provide relatively *weak privacy* guarantees: every bit that each phone transmits leaks some private user information to the vendor. In particular, when $p = 0.1$ the vendor has a good chance of seeing the correct (unflipped) user response. Increasing the noise level p decreases this leakage, but adding more noise also decreases the accuracy of the vendor's final estimate. As an example, assume that the vendor collects randomized responses from one million phones using $p = 0.49$, and that 1% of phones have the sensitive app installed. Even with such a large number of responses, the vendor will incorrectly conclude that *no phones* have the app installed roughly one third of the time.

An alternative approach to the data-collection problem is to have the phones send *encryptions* of their bits to a set of servers. The servers can sum up the encrypted bits and decrypt only the final sum [107, 119, 176, 207, 230, 231]. As long as all servers do not collude, these encryption-based systems provide much *stronger privacy* guarantees: the system leaks nothing about a user's private bit to the vendor, except what the vendor can infer from the final sum. By carefully adding structured noise to the final sum, these systems can provide differential privacy as well [119, 207, 249].

However, in gaining this type of privacy, many secret-sharing-based systems sacrifice *robustness*: a malicious client can send the servers an encryption of a large integer value v instead of a zero/one bit. Since the client's value v is encrypted, the servers cannot tell from inspecting the ciphertext that $v > 1$. Using this approach, a single malicious client can increase the final sum by v, instead of by 1. Clients often have an incentive to cheat in this way: an app developer could use this attack to boost the perceived popularity of her app, with the goal of getting it to appear on the app store's home page. It is possible to protect against these attacks using traditional zero-knowledge proofs [249], but these protections destroy concrete *efficiency*: checking the proofs requires heavy public-key cryptographic operations at the servers and can increase the servers' workload by orders of magnitude.

In this paper, we introduce Prio, a system for private aggregation that resolves the tension between privacy, robustness, and efficiency. Prio uses a small number of servers; as long as one of the Prio servers is honest, the system leaks nearly nothing about clients' private data (in a sense we precisely define), except what the aggregate statistic itself reveals. In this sense, Prio provides a strong form of cryptographic *privacy*. This property holds even against an adversary who can observe the entire network, control all but one of the servers, and control a large number of clients.

Prio also maintains *robustness* in the presence of an unbounded number of malicious clients, since the Prio servers can detect and reject syntactically incorrect client submissions in a privacy-preserving way. For instance, a car cannot report a speed of 100,000 km/h if the system parameters only allow speeds between 0 and 200 km/h. Of course, Prio cannot prevent a malicious client from submitting an untruthful data value: for example, a faulty car can always misreport its actual speed.

To provide robustness, Prio applies the technique of zero-knowledge proofs on secret-shared data, which we developed in Section 3.3. When a client sends an encoding of its private data to the Prio servers, the client also sends to each server a "share" of a zero-knowledge proof of correctness. Even if the client is malicious and the proof shares are malformed, the servers can use these shares to collaboratively check—without ever seeing the client's private data in the clear—that the client's encoded submission is syntactically valid. Our zero-knowledge proofs on secret-shared data rely only upon fast, information-theoretic cryptography, and concretely require the servers to exchange only a few hundred bytes of information to check each client's submission.

Prio provides privacy and robustness without sacrificing concrete *efficiency*. When deployed on a collection of five servers spread around the world and configured to compute private sums over vectors of private client data, Prio imposes a 5.7× slowdown over a naïve data-collection system that provides no privacy guarantees whatsoever. In contrast, a state-of-the-art comparison system that uses client-generated non-interactive discrete-log-based zero-knowledge proofs of correctness (NIZKs) [50, 241] imposes a 267× slowdown at the servers. Prio improves client performance as well: it is 50-100× faster than NIZKs and we estimate that it is three orders of magnitude faster than methods based on succinct non-interactive arguments of knowledge (SNARKs) [37, 134, 225]. The system is fast in absolute terms as well: when configured up to privately collect the distribution of responses to a survey with over 400 true/false questions, the client performs only 26 ms of computation,

and our distributed cluster of Prio servers can process each client submission in under 2 ms on average.

Contributions. In this work, we:

- apply our zero-knowledge proofs on secret-shared data (Section 3.3) to solve a practical privacy problem,
- present *affine-aggregatable encodings*, a framework that unifies many data-encoding techniques used in prior work on private aggregation, and
- demonstrate how to combine these encodings with proofs on secret-shared data to provide robustness and privacy in a large-scale data-collection system.

With Prio, we demonstrate that data-collection systems can simultaneously achieve strong privacy, robustness to faulty clients, and performance at scale.

Deployment in Firefox. As we were working on the Prio system, we learned that Mozilla, the company behind the Firefox browser, was looking for a privacy-preserving way to collect telemetry data from its users. After considering alternative systems, which either provided weaker privacy or performance properties, Mozilla chose to use Prio to as the basis for its new privacy-preserving telemetry system. Working with Mozilla engineers, we wrote libprio (available at https://github.com/mozilla/libprio), an optimized Prio implementation in C, and helped tie it into the Firefox browser. This library now ships to millions of Firefox users. In Section 1.2, we give more details on the deployment of Prio in Firefox.

4.2 System goals

A Prio deployment consists of a small number of infrastructure servers and a very large number of clients. In each time epoch, every client i in the system holds a private value x_i. The goal of the system is to allow the servers to compute $f(x_1, \ldots, x_n)$, for some aggregation function f, in a way that leaks as little as possible about each client's private x_i values to the servers.

Informally, Prio protects client *privacy* as long as at least one server is honest; Prio provides *robustness* (correctness) only if all servers are honest.

System model. The parties to a Prio deployment must establish pairwise authenticated and encrypted channels. Towards this end, we assume the existence of a public-key infrastructure

and the basic cryptographic primitives (CCA-secure public-key encryption [99, 250, 251], digital signatures [152], etc.) that make secure channels possible. We make no synchrony assumptions about the network: the adversary may drop or reorder packets on the network at will, and the adversary may monitor all links in the network. Low-latency anonymity systems, such as Tor [112], provide no anonymity in this setting, and Prio does not rely on such systems to protect client privacy.

Goal 1: Privacy

For an aggregation function f, we say that Prio provides f-privacy, if an adversary that controls any number of clients and all but one server learns nothing about the honest clients' values x_i, except what she can infer from the value $f(x_1, \ldots, x_n)$ itself. More precisely, given $f(x_1, \ldots, x_n)$, every adversary controlling a proper subset of the servers, along with any number of clients, can simulate its view of the protocol execution.

In what follows we use the standard notions of *negligible functions* and *computational indistinguishability* (see, e.g., [144]). We often leave the security parameter implicit.

Definition 4.2.1 (f-Privacy)**.** For an s-server Prio deployment with n clients, we say that the scheme provides *f-privacy* for a function f if for:

- every number $t \leq s - 1$ of malicious servers, and
- every number of malicious clients $m \leq n$,

there exists an efficient simulator that, for every choice of the honest clients' inputs $(x_1, \ldots, x_{n-m})$, takes as input:

- the public parameters to the protocol run (all participants' public keys, the description of the aggregation function f, the cryptographic parameters, etc.),
- the indices of the adversarial clients and servers,
- oracle access to the adversarial participants, and
- the value $f(x_1, \ldots, x_{n-m})$,

and outputs a simulation of the adversarial participants' view of the protocol run whose distribution is computationally indistinguishable from the distribution of the adversary's view of the real protocol run.

By engaging in the protocol, the adversary learns the value $f(x_1, \ldots, x_{n-m})$ exactly. It is therefore critical that the honest servers ensure that the number $n - m$ of honest clients'

values included in the aggregate is "large enough." The honest servers must use out-of-band means to ensure that many honest clients' values are included in the final aggregate. See discussion of this and related issues in Section 4.9.

Prio does not natively provide differential privacy [115], since the system adds no noise to the aggregate statistics it computes. In Section 4.9, we discuss when differential privacy may be useful and how we can extend Prio to provide it.

For many of the aggregation functions f that Prio implements, Prio provides strict f-privacy. For some aggregation functions, which we highlight in Section 4.5, Prio provides $\hat{f}$-privacy, where $\hat{f}$ is a function that outputs slightly more information than f. More precisely, $\hat{f}(x_1, \ldots, x_n) = \langle f(x_1, \ldots, x_n),\ L(x_1, \ldots, x_n) \rangle$ for some modest leakage function L.

Goal 2: Anonymity

A data-collection scheme maintains client anonymity if the adversary cannot tell which honest client submitted which data value through the system, even if the adversary chooses the honest clients' data values, controls all other clients, and controls all but one server. Prio always protects client anonymity.

More formally, let SORT be the function that takes n inputs and outputs them in lexicographic order.

Definition 4.2.2 (Anonymity)**.** We say that a data-collection scheme provides *anonymity* if it provides f-privacy, in the sense of Definition 4.2.1, for $f = \mathsf{SORT}$.

A scheme that provides this form of anonymity leaks to the adversary the entire list of honest clients' inputs $(x_1, \ldots, x_{n-m})$, but the adversary learns nothing about which client submitted which value x_i. For example, if each client submits their location via a data-collection scheme that provides anonymity, the servers learn the list of submitted locations $\{\ell_1, \ldots, \ell_{n-m}\}$, but the servers learn nothing about whether a particular honest client is in a particular location ℓ^*.

We now briefly argue that Prio always provides anonymity. First, we assert that Prio can only compute aggregate statistics for *symmetric* aggregation functions f, where the notion of symmetric functions is as follows. (The truth of the assertion follows by inspection of the construction in Section 4.3.) Next, we prove that any scheme that provides f-privacy, for a symmetric function f, also provides anonymity.

Definition 4.2.3 (Symmetric function)**.** A function $f(x_1, \ldots, x_n)$ is *symmetric* if, for all permutations π on n elements, the equality $f(x_1, \ldots, x_n) = f(x_{\pi(1)}, \ldots, x_{\pi(n)})$ holds.

Claim 4.2.4. *Let $\mathcal{D}$ be a data-collection scheme that provides f-privacy, in the sense of Definition 4.2.1, for a symmetric function f. Then $\mathcal{D}$ provides anonymity.*

Proof sketch. The fact that $\mathcal{D}$ provides f-privacy implies the existence of a simulator $S_{\mathcal{D}}$ that takes as input $f(x_1, \ldots, x_{n-m})$, along with other public values, and induces a distribution of protocol transcripts indistinguishable from the real one. If f is symmetric, $f(x_1, \ldots, x_{n-m}) = f(x'_1, \ldots, x'_{n-m})$, where

$$(x'_1, \ldots, x'_{n-m}) = \mathsf{SORT}(x_1, \ldots, x_{n-m}).$$

Using this fact, we construct the simulator required for the anonymity definition: on input $(x'_1, \ldots, x'_{n-m}) = \mathsf{SORT}(x_1, \ldots, x_{n-m})$, compute $f(x'_1, \ldots, x'_{n-m})$, and feed the output of f to the simulator $S_{\mathcal{D}}$. The validity of the simulation is immediate. □

The following claim demonstrates that it really only makes sense to use an f-private data collection scheme when the function f is symmetric, as all of the functions we consider in Prio are.

Claim 4.2.5. *Let f be a non-symmetric function on any number of inputs n. Then there is no anonymous data collection scheme that correctly computes f.*

Proof sketch. Let $n = 2$ and let there be no malicious clients ($m = 0$). Because f is not symmetric, there must exist an input (x_1, x_2) in the domain of f such that $f(x_1, x_2) \neq f(x_2, x_1)$.

Let $\mathcal{D}$ be a data-collection scheme that implements the aggregation function $f(x_1, x_2)$. This $\mathcal{D}$ outputs $f(x_1, x_2)$ for all x_1, x_2 in the domain of f, and hence $f(x_1, x_2)$ is part of the protocol transcript.

For $\mathcal{D}$ to be anonymous, there must be a simulator that takes $\mathsf{SORT}(x_1, x_2)$ as input, and simulates the protocol transcript. In particular, it must output $f(x_1, x_2)$. But given $\mathsf{SORT}(x_1, x_2)$, the simulator has no information (in a computational sense) on whether the parties' inputs were (x_1, x_2) or (x_2, x_1). Thus, the simulator can do no better than to guess which of these cases it is in, and the simulation will fail with noticeable probability. □

Goal 3: Robustness

A private aggregation system is robust if a coalition of malicious clients can affect the output of the system only by misreporting their private data values; a coalition of malicious clients cannot otherwise corrupt the system's output. For example, if the function $f(x_1, \ldots, x_n)$ counts the number of times a certain string appears in the set $\{x_1, \ldots, x_n\}$, then a single malicious client should not be able to affect the count by more than one.

Prio is robust only against adversarial clients—*not* against adversarial servers. Although providing robustness against malicious servers seems desirable at first glance, doing so would come at privacy and performance costs, which we discuss in Remark 4.2.7. Since there could be millions of clients in a Prio deployment, and only a handful of servers (in fixed locations with known administrators), it may also be possible to eject faulty servers using out-of-band means.

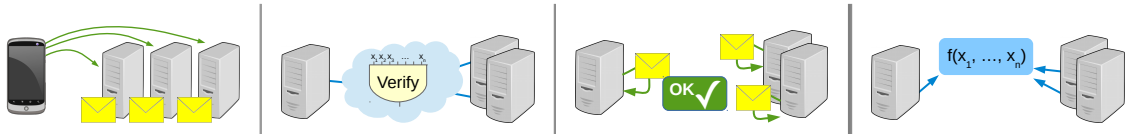
More formally, recall that each Prio client holds a value x_i, where the value x_i is an element of some set of data items $\mathcal{D}$. For example, $\mathcal{D}$ might be the set of 4-bit integers. The definition of robustness states that when all servers are honest, a set of malicious clients cannot influence the final aggregate, beyond their ability to choose arbitrary *valid* inputs. For example, malicious clients can choose arbitrary 4-bit integers as their input values, but cannot influence the output in any other way.

Definition 4.2.6 (Robustness)**.** Fix a security parameter $\lambda > 0$. We say that an n-client Prio deployment provides *robustness* if, when all Prio servers execute the protocol faithfully, for every number m of malicious clients (with $0 \leq m \leq n$), and for every choice of honest client's inputs $(x_1, \ldots, x_{n-m}) \in \mathcal{D}^{n-m}$, the servers, with all but negligible probability in λ, output a value in the set:

$$\Big\{ f(x_1, \ldots, x_n) \mid (x_{n-m+1}, \ldots, x_n) \in \mathcal{D}^m \Big\}.$$

Remark 4.2.7 (Robustness against faulty servers). If at least one of the servers is honest, Prio ensures that the adversary learns nothing about clients' data, except the aggregate statistic. In other words, Prio ensures *client privacy* if at least one of servers is honest.

Prio provides *robustness* only if all servers are honest. Providing robustness in the face of faulty servers is obviously desirable, but we are not convinced that it is worth the security and performance costs. First, providing robustness necessarily weakens the privacy guarantees that the system provides: if the system protects *robustness* in the presence of t faulty servers,



(a) The client sends a share of its encoded submission and zero-knowledge proof to each server.

(b) The servers validate the client's proof to ensure that the submission is valid.

(c) If the checks pass, the servers update their local accumulators with the client-provided data.

(d) After accumulating many packets, the servers publish their accumulators to reveal the aggregate.

Figure 4.1: An overview of the Prio pipeline for processing client submissions.

out of s servers total, then the system can protect *privacy* only against a coalition of at most $s-t-1$ malicious servers. The reason is that, if robustness holds against t faulty servers, then $s-t$ honest servers must be able to produce a correct output even if these t faulty servers are offline. Put another way: $s-t$ *dishonest* servers can recover the output of the system even without the participation of the t honest servers. Instead of computing an aggregate over many clients ($f(x_1, \ldots, x_n)$), the dishonest servers can compute the "aggregate" over a single client's submission ($f(x_1)$) and essentially learn that client's private data value.

So strengthening robustness in this setting weakens privacy. Second, protecting robustness comes at a performance cost: some our optimizations use a "leader" server to coordinate the processing of each client submission (see Section 4.7). A faulty leader cannot compromise privacy, but *can* compromise robustness. Strengthening the robustness property would force us to abandon these optimizations.

That said, it would be possible to extend Prio to provide robustness in the presence of corrupt servers using standard techniques [25] (replace s-out-of-s secret sharing with Shamir's threshold secret-sharing scheme [247], etc.).

4.3 A simple scheme

Let us introduce Prio by first presenting a simplified version of it. In this simple version, each client holds a one-bit integer x_i and the servers want to compute the sum of the clients' private values $\sum_{i=1}^{n} x_i$. Even this very basic functionality has many real-world applications. For example, the developer of a health data mobile app could use this scheme to collect the number of app users who have a certain medical condition. In this application, the bit x_i would indicate whether the user has the condition, and the sum over the x_is gives the count of affected users.

The public parameters for the Prio deployment include the description of a finite field $\mathbb{F}$ (e.g., as specified by a prime p). The simplified s-server Prio scheme for computing sums proceeds in three steps.

1. **Upload.** Each client i splits its private bit $x_i \in \{0, 1\}$ into s shares, one per server, using a secret-sharing scheme. In particular, the client picks random integers $[\![x]\!]_1, \ldots, [\![x]\!]_s \in \mathbb{F}$, subject to the constraint: $x_i = [\![x]\!]_1 + \cdots + [\![x]\!]_s \in \mathbb{F}$. The client then sends, over an encrypted and authenticated channel, one share of its submission to each server.
2. **Aggregate.** Each server j holds an accumulator value $A_j \in \mathbb{F}$, initialized to zero. Upon receiving a share from the ith client, the server adds the uploaded share into its accumulator: $A_j \leftarrow A_j + [\![x]\!]_j \in \mathbb{F}$.
3. **Publish.** Once the servers have received a share from each client, they publish their accumulator values. Computing the sum of the accumulator values $\sum_{j=1}^{s} A_j \in \mathbb{F}$ yields the desired sum $\sum_{i=1}^{n} x_i$ of the clients' private values, as long as the characteristic of the field $\mathbb{F}$ is larger than the number of clients. When working in $\mathbb{F}_p$ (i.e., modulo a prime p), this condition is equivalent to saying that the sum $\sum_{i=1}^{n} x_i \bmod p$ does not "overflow" the modulus.

There are two observations we can make about this scheme. First, even this simple scheme provides privacy: the servers learn the sum $\sum_{i=1}^{n} x_i$ but they learn nothing else about the clients' private inputs. Second, the scheme does *not* provide robustness. A single malicious client can completely corrupt the protocol output by submitting (for example), a random integer $r \in \mathbb{F}$ to each server.

The core contributions of Prio are to improve this basic scheme in terms of security and functionality. In terms of security, Prio extends the simple scheme to provide robustness in the face of malicious clients. In terms of functionality, Prio extends the simple scheme to allow privacy-preserving computation of a wide array of aggregation functions (not just sum).

4.4 Protecting robustness with proofs on secret-shared data

Upon receiving shares of a client's data value, the Prio servers need a way to check if the client-submitted value is well formed. For example, in the simplified protocol of Section 4.3, every client is supposed to send the servers the share of a value x such that $0 \leq x \leq 1$. However,

since the client sends only a *single share* of its value x to each server—to preserve privacy—each server essentially receives an encoded version of x and cannot unilaterally determine if x is well formed. In the more general setting, each Prio client submits to each server a share $[\![x]\!]_j$ of a vector $x \in \mathbb{F}^k$, for some finite field $\mathbb{F}$. The servers hold a validation predicate $\mathsf{Valid}(\cdot)$, and should only accept the client's data submission if $\mathsf{Valid}(x) = 1$ (Figure 4.1).

To execute this check in Prio, we apply the new cryptographic tool of *zero-knowledge proofs on secret-shared data*, introduced in Section 3.3. With these proofs, the client can quickly prove to the servers that $\mathsf{Valid}(x) = 1$, for an arbitrary function Valid, without leaking anything else about its private input x to the servers.

4.4.1 Overview

While Section 3.3 discusses zero-knowledge proofs on secret-shared data in detail, here we recall the relevant properties needed for our application.

A (one-round) zero-knowledge proof on secret-shared data consists of an interaction between a client (the prover) and multiple servers (the verifiers). At the start of the protocol:

- each server j holds a vector $[\![x]\!]_j \in \mathbb{F}^k$,
- the client holds the vector $x = \sum_{j=1}^{s} [\![x]\!]_j \in \mathbb{F}^k$, and
- all parties hold an arithmetic circuit representing a predicate $\mathsf{Valid} : \mathbb{F}^k \to \mathbb{F}$.

The client's goal is to convince the servers that $\mathsf{Valid}(x) = 1$, without leaking anything else about x to the servers. To do so, the client sends a proof string to each server. After receiving these proof strings, the servers gossip amongst themselves and then conclude either that $\mathsf{Valid}(x) = 1$ (the servers "accept x") or not (the servers "reject x").

For a zero-knowledge proof on secret-shared data to be useful in Prio, it must satisfy the following properties:

Correctness. If all parties are honest, the servers will accept x.

Soundness. If all servers are honest, and if $\mathsf{Valid}(x) \neq 1$, then for all malicious clients, even ones running in super-polynomial time, the servers will reject x with overwhelming probability. In other words, no matter how the client cheats, the servers will almost always reject x.

Zero knowledge. If the client and at least one server are honest, then the servers learn nothing about x, except that $\mathsf{Valid}(x) = 1$. More precisely, there exists a simulator

(that does not take x as input) that accurately reproduces the view of any proper subset of malicious servers executing the verification protocol.

We formally defined zero-knowledge proofs on distributed data in Chapter 3. See Section 3.3 for discussion of the special case of proofs on secret-shared data.

Construction. The zero-knowledge proof on secret-shared data that we use in Prio requires minimal server-to-server communication, is compatible with any public Valid circuit, and relies solely on fast, information-theoretic primitives. (We discuss how to hide the Valid circuit from the client in Section 4.4.2.)

We construct this proof on secret-shared data using the tools of Chapter 2. First, we take the fully linear PCP of Theorem 2.3.3, instantiated with a G-gate defined as $G(y, z) \stackrel{\text{def}}{=} y \cdot z \in \mathbb{F}$. This yields a fully linear PCP for an arithmetic circuit Valid that has proof length |Valid| field elements, query complexity $O(1)$, and strong honest-verifier zero knowledge. Next, we "compile" this fully linear PCP into a zero-knowledge proof on secret-shared data using the compiler of Theorem 3.2.1. This yields a zero-knowledge proof on secret-shared data that protects zero knowledge even against malicious servers. The proof system requires the client to send each server one message of length |Valid| field elements. The servers exchange a constant number of field elements to check the proof.

In the current implementation of Prio, the underlying fully linear PCP does *not* have zero knowledge, so we apply the idea of Section 3.3.1 to add zero knowledge to the proof system on secret-shared data. By carefully exploiting the structure of the fully linear PCP, we ensure that the resulting proof on secret-shared data still maintains zero knowledge against malicious verifiers. (We plan to modify the underlying linear PCP to have zero-knowledge to avoid this extra step.)

Efficiency. The notable property of these zero-knowledge proofs on secret-shared data is that the server-to-server communication cost grows neither with the complexity of the verification circuit nor with the size of the value x (Table 4.1). The computation cost at the servers is essentially the same as the cost for each server to evaluate the Valid circuit locally.

That said, the client-to-server communication cost does grow linearly with the size of the Valid circuit, provided that we allow the Valid circuit to be an arbitrary arithmetic circuit. When the Valid circuit has repeated structure (e.g., the circuit checks whether the client has uploaded a large vector of 4-bit integers), we can apply more sophisticated proof systems from Chapter 2, to reduce the client's bandwidth usage.

		NIZK	SNARK	**Prio**
Client	Exponentiations	k	n	0
	Multiplications in $\mathbb{F}$	0	$k \log k$	$k \log k$
	Proof length	k	1	k
Servers	Exponentiatons/Pairings	k	1	0
	Multiplications in $\mathbb{F}$	0	k	$k \log k$
	Data transfer	k	1	1

Table 4.1: An asymptotic comparison of Prio with standard zero-knowledge techniques showing that Prio reduces the computational burden for clients and servers. The client holds a vector $x \in \mathbb{F}^k$, each server i holds a share $[\![x]\!]_i$, and the client convinces the servers that each component of x is a 0/1 value in $\mathbb{F}$. For readability, we suppress the $\Theta(\cdot)$ notation and fixed polynomials in the security parameter. We measure proof length and communication in terms of $\mathbb{F}$ elements.

4.4.2 Hiding the validity predicate

Constructing the zero-knowledge proof requires the client to compute $\mathsf{Valid}(x)$ on its own. If the verification circuit takes secret server-provided values as input, or is itself a secret belonging to the servers, then the client does not have enough information to compute $\mathsf{Valid}(x)$ on its own. For example, the servers could run a proprietary verification algorithm to detect spammy client submissions—the servers would want to run this algorithm without revealing it to the (possibly spam-producing) clients. To handle this use case, the servers can execute the verification check themselves at a slightly higher cost.

Crucially, this server-side variant only provides security against "honest-but-curious" servers, who execute the protocol correctly but who try to learn everything possible about the clients' private data from the protocol execution. In contrast, the variant based on our proofs on secret-shared data maintains privacy against actively malicious servers. It may be possible to provide security against actively malicious servers using a client-assisted variant of the SPDZ multi-party computation protocol [103], though we leave this extension to future work.

Our idea works in three steps:

1. The client provides correlated randomness to the servers, along with a proof on secret-shared data asserting that this randomness is well formed.
2. The servers verify the proof on secret-shared data to confirm that the randomness is well formed.

3. The servers use the correlated randomness to run a secure multiparty computation to compute the value $\mathsf{Valid}(x)$ without learning anything about the client's private input x.

Our idea is to rely on a classic result of Beaver [24], which implies that if the servers hold additive shares of M triples $\left\{(a_t, b_t, c_t) \in \mathbb{F}^3\right\}_{t=1}^{M}$, such that $a_t \cdot b_t = c_t \in \mathbb{F}$ for all $t \in \{1, \ldots, M\}$, the servers can securely compute any arithmetic circuit over $\mathbb{F}$ that contains at most M multiplication gates. As we use it, Beaver's protocol only provides security when the servers follow the protocol honestly (i.e., are "honest but curious").

Let M be the number of multiplication gates in the Valid circuit. So, the client first generates M multiplication triples, splits these into additive shares, and sends one set of shares to each server, along with a share of its private value x.

Let the t-th multiplication triple be of the form $(a_t, b_t, c_t) \in \mathbb{F}^3$. Then, client can use a zero-knowledge proof on secret-shared data (Section 4.4.1) to convince the servers that all of the M triples it sent the servers are well-formed. In particular, the client proves to the server—who hold additive shares of the M triples—that $c_t = a_t \cdot b_t$, for all $t \in \{1, \ldots, M\}$.

After the servers verify the proof, they can execute Beaver's multiparty computation protocol to evaluate the circuit using the M client-provided multiplication triples. The security of this portion of the protocol, with respect to "honest-but-curious" servers, follows directly from the security of Beaver's protocol.

Running the computation requires the servers to exchange $\Theta(M)$ field elements, and the number of rounds of communication is proportional to the multiplicative depth of the Valid circuit (i.e., the maximum number of multiplication gates on an input-output path).

4.5 Gathering complex statistics

So far, we have developed the means to compute private sums over client-provided data (Section 4.3) and to check an arbitrary validation predicate against secret-shared data (Section 4.4). Combining these two ideas with careful data encodings, which we introduce now, allows Prio to compute more sophisticated statistics over private client data.

At a high level, each client first encodes its private data value in a prescribed way, and the servers then privately compute the sum of the encodings. Finally, the servers can decode the summed encodings to recover the statistic of interest. The participants perform this encoding and decoding via a mechanism we call affine-aggregatable encodings ("AFEs").

4.5.1 Overview of affine-aggregatable encodings (AFEs)

In our setting, each client i holds a value $x_i \in \mathcal{D}$, where $\mathcal{D}$ is some set of data values. The servers hold an aggregation function $f : \mathcal{D}^n \to \mathcal{A}$, whose range is a set of aggregates $\mathcal{A}$. For example, the function f might compute the standard deviation of its n inputs. The servers' goal is to evaluate $f(x_1, \ldots, x_n)$ without learning the x_is.

An AFE gives an efficient way to encode the data values x_i such that it is possible to compute the value $f(x_1, \ldots, x_n)$ given only the *sum of the encodings* of $x_1, \ldots, x_n$. An AFE consists of three efficient algorithms $(\mathsf{Encode}, \mathsf{Valid}, \mathsf{Dec})$, defined with respect to a field $\mathbb{F}$ and two integers k and k', where $k' \leq k$:

- $\mathsf{Encode}(x)$: maps an input $x \in \mathcal{D}$ to its encoding in $\mathbb{F}^k$,
- $\mathsf{Valid}(y)$: returns true if and only if $y \in \mathbb{F}^k$ is a valid encoding of some data item in $\mathcal{D}$,
- $\mathsf{Dec}(\sigma)$: takes $\sigma = \sum_{i=1}^{n} \mathsf{Trunc}_{k'}\big(\mathsf{Encode}(x_i)\big) \in \mathbb{F}^{k'}$ as input, and outputs $f(x_1, \ldots, x_n)$. The $\mathsf{Trunc}_{k'}(\cdot)$ function outputs the first $k' \leq k$ components of its input.

The AFE uses all k components of the encoding in validation, but only uses k' components to decode σ. In many of our applications we have $k' = k$.

An AFE is *private with respect to a function* $\hat{f}$, or simply $\hat{f}$-private, if σ reveals nothing about $x_1, \ldots, x_n$ beyond what $\hat{f}(x_1, \ldots, x_n)$ itself reveals. More precisely, it is possible to efficiently simulate σ given only $\hat{f}(x_1, \ldots, x_n)$. Usually $\hat{f}$ reveals nothing more than the aggregation function f (i.e., the minimum leakage possible), but in some cases $\hat{f}$ reveals a little more than f.

For some functions f we can build more efficient f-private AFEs by allowing the encoding algorithm to be randomized. In these cases, we allow the decoding algorithm to return an answer that is only an approximation of f, and we also allow it to fail with some small probability.

Prior systems have made use of specific AFEs for sums [119, 188], standard deviations [231], counts [65, 207], and least-squares regression [179]. Our contribution is to unify these notions and to adopt existing AFEs to enable better composition with Prio's zero-knowledge proofs on secret-shared data. In particular, by using more complex encodings, we can reduce the size of the circuit for Valid, which results in shorter proofs.

AFEs in Prio: Putting it all together. The full Prio system computes $f(x_1, \ldots, x_n)$ privately as follows (see Figure 4.1): Each client encodes its data value x using the AFE Encode routine for the aggregation function f. Then, as in the simple scheme of Section 4.3,

every client splits its encoding into s shares and sends one share to each of the s servers. The client uses a proof on secret-shared data (Section 4.4) to convince the servers that its encoding satisfies the AFE Valid predicate.

Upon receiving a client's submission, the servers verify the proof to ensure that the encoding is well-formed. If the servers conclude that the encoding is valid, every server adds the first k' components of the encoding share to its local running accumulator. (Recall that k' is a parameter of the AFE scheme.) Finally, after collecting valid submissions from many clients, every server publishes its local accumulator, enabling anyone to run the AFE Dec routine to compute the final statistic in the clear. The formal description of the system is presented in Section 4.6, where we also analyze its security.

Limitations. There exist aggregation functions for which all AFE constructions must have large encodings. For instance, say that each of n clients holds an integer x_i, where $1 \leq x_i \leq n$. We might like an AFE that computes the median of these integers $\{x_1, \ldots, x_n\}$, working over a field $\mathbb{F}$ with $|\mathbb{F}| \approx n^d$, for some constant $d \geq 1$.

We show that there is no such AFE whose encodings consist of $n' \in o(n/\log n)$ field elements. Suppose, towards a contradiction, that such an AFE did exist. Then we could describe any sum of encodings using at most $O(n' \log |\mathbb{F}|) = o(n)$ bits of information. From this AFE, we could build a single-pass, space-$o(n)$ streaming algorithm for computing the exact median of an n-item stream. But every single-pass streaming algorithm for computing the exact median over an n-item stream requires $\Omega(n)$ bits of space [160], which is a contradiction. Similar arguments may rule out space-efficient AFE constructions for other natural functions.

4.5.2 Definition of affine-aggregatable encodings

An AFE is defined relative to a field $\mathbb{F}$, two integers k and k' (where $k' \leq k$), a set $\mathcal{D}$ of data elements, a set $\mathcal{A}$ of possible values of the aggregate statistic, and an aggregation function $f : \mathcal{D}^n \to \mathcal{A}$. An AFE scheme consists of three efficient algorithms. The algorithms are:

- $\mathsf{Encode} : \mathcal{D} \to \mathbb{F}^n$. Covert a data item into its AFE-encoded counterpart.
- $\mathsf{Valid} : \mathbb{F}^n \to \{0, 1\}$. Return "1" if and only if the input is in the image of Encode.
- $\mathsf{Dec} : \mathbb{F}^{k'} \to \mathcal{A}$. Given a vector representing a collection of encoded data items, return the value of the aggregation function f evaluated at these items.

To be useful, an AFE encoding should satisfy the following properties:

Definition 4.5.1 (AFE correctness)**.** We say that an AFE is *correct* for an aggregation function f if, for every choice of $(x_1, \ldots, x_n) \in \mathcal{D}^n$, we have that:

$$\mathsf{Dec}\big(\textstyle\sum_{i=1}^{n} \mathsf{Trunc}_{k'}\big(\mathsf{Encode}(x_i)\big)\ \big) = f(x_1, \ldots, x_n).$$

Recall that $\mathsf{Trunc}_{k'}(v)$ denotes truncating the vector $v \in \mathbb{F}^k$ to its first k' components.

The correctness property of an AFE essentially states that if we are given valid encodings of data items $(x_1, \ldots, x_n) \in \mathcal{D}^n$, the decoding of their sum should be $f(x_1, \ldots, x_n)$.

Definition 4.5.2 (AFE soundness)**.** We say that an AFE is *sound* if, for all encodings $e \in \mathbb{F}^k$: the predicate $\mathsf{Valid}(e) = 1$ if and only if there exists a data item $x \in \mathcal{D}$ such that $e = \mathsf{Encode}(x)$.

An AFE is private with respect to a function $\hat{f}$, if the sum of encodings

$$\sigma = \sum_{i=1}^{n} \mathsf{Trunc}_{k'}(\mathsf{Encode}(x_i)),$$

given as input to algorithm Dec, reveals nothing about the underlying data beyond what $\hat{f}(x_1, \ldots, x_n)$ reveals.

Definition 4.5.3 (AFE privacy)**.** We say that an AFE is *private* with respect to a function $\hat{f} : \mathcal{D}^n \to \mathcal{A}$ if there exists an efficient simulator S such that for all input data $(x_1, \ldots, x_n) \in \mathcal{D}^n$, the distribution $S(\hat{f}(x_1, \ldots, x_n))$ is indistinguishable from the distribution $\sigma = \sum_{i=1}^{n} \mathsf{Trunc}_{k'}(\mathsf{Encode}(x_i))$.

Relaxed correctness. In many cases, randomized data structures are more efficient than their deterministic counterparts. We can define a relaxed notion of correctness to capture a correctness notion for randomized AFEs. In the randomized case, the scheme is parameterized by constants $0 < \delta, \epsilon$ and the Dec algorithm may use randomness. We demand that with probability at least $1 - 2^{-\delta}$, over the randomness of the decoding algorithms, the encoding yields an "ϵ-good" approximation of f. In our applications, typically an ϵ-good approximation is within a multiplicative or additive factor of ϵ from the true value; the exact meaning depends on the AFE in question.

4.5.3 Aggregating basic data types

This section presents the basic affine-aggregatable encoding schemes that serve as building blocks for the more sophisticated schemes. In the following constructions, the clients hold data values $x_1, \ldots, x_n \in \mathcal{D}$, and our goal is to compute an aggregate $f(x_1, \ldots, x_n)$.

In constructing these encodings, we have two goals. The first is to ensure that the AFE leaks as little as possible about the x_is, apart from the value $f(x_1, \ldots, x_n)$ itself. The second is to minimize the number of multiplication gates in the arithmetic circuit for Valid, since the cost of the zero-knowledge proofs on secret-shared data grows with the circuit size.

In what follows, we let λ be a security parameter, such as $\lambda = 80$ or $\lambda = 128$.

Integer sum and mean. We first construct an AFE for computing the sum of b-bit integers. Let $\mathbb{F}$ be a finite field of size at least $n2^b$. On input $0 \leq x \leq 2^b - 1$, the $\mathsf{Encode}(x)$ algorithm first computes the bit representation of x, denoted $(\beta_0, \beta_1, \ldots, \beta_{b-1}) \in \{0,1\}^b$. It then treats the binary digits as elements of $\mathbb{F}$, and outputs

$$\mathsf{Encode}(x) = (x, \beta_0, \ldots, \beta_{b-1}) \in \mathbb{F}^{b+1}.$$

To check that x represents a b-bit integer, the Valid algorithm ensures that each β_i is a bit, and that the bits represent x. Specifically, the algorithm checks that the following equalities hold over $\mathbb{F}$:

$$\mathsf{Valid}(\mathsf{Encode}(x)) = \left(x = \sum_{i=0}^{b-1} 2^i \beta_i\right) \wedge \bigwedge_{i=0}^{b-1} \Big[(\beta_i - 1) \cdot \beta_i = 0\Big].$$

The Dec algorithm takes the sum of encodings σ as input, truncated to only the first coordinate. That is, $\sigma = \sum_{i=1}^{n} \mathsf{Trunc}_1\big(\mathsf{Encode}(x_1)\big) = x_1 + \cdots + x_n$. This σ is the required aggregate output. Moreover, this AFE is clearly sum-private.

To compute the arithmetic mean, we divide the sum of integers by n over the rationals. Computing the product and geometric mean works in exactly the same matter, except that we encode x using b-bit logarithms.

Variance and STDDEV. Using known techniques [72, 231], the summation AFE above lets us compute the variance of a set of b-bit integers using the identity: $\mathrm{Var}(X) = \mathrm{E}[X^2] - (\mathrm{E}[X])^2$. Each client encodes its integer x as (x, x^2) and then applies the summation AFE to each of

the two components. (The Valid algorithm also ensures that second integer is the square of the first.) The resulting two values let us compute the variance.

This AFE also reveals the expectation $\mathrm{E}[X]$. It is private with respect to the function $\hat{f}$ that outputs both the expectation and variance of the given set of integers.

Boolean OR and AND. When $\mathcal{D} = \{0, 1\}$ and $f(x_1, \ldots, x_n) = \text{OR}(x_1, \ldots, x_n)$ the encoding operation outputs an element of $\mathbb{F}_2^\lambda$ (i.e., a λ-bit bitstring) as:

$$\mathsf{Encode}(x) = \begin{cases} \lambda \text{ zeros} & \text{if } x = 0 \\ \text{a random element in } \mathbb{F}_2^\lambda & \text{if } x = 1. \end{cases}$$

The Valid algorithm outputs "1" always, since all λ-bit encodings are valid. The sum of encodings is simply the XOR of the n λ-bit encodings. The Dec algorithm takes as input a λ-bit string and outputs "0" if and only if its input is a λ-bit string of zeros. With probability $1 - 2^{-\lambda}$, over the randomness of the encoding algorithm, the decoding operation returns the boolean OR of the encoded values. This AFE is OR-private. A similar construction yields an AFE for boolean AND.

MIN and MAX. To compute the minimum and maximum of integers over a range $\{0, \ldots, B-1\}$, where B is small (e.g., car speeds in the range 0–250 km/h), the Encode algorithm can represent each integer in unary as a length-B vector of bits $(\beta_0, \ldots, \beta_{B-1})$, where $\beta_i = 1$ if and only if the client's value $x \leq i$. We can use the bitwise-OR construction above to take the OR of the client-provided vectors—the largest value containing a "1" is the maximum. To compute the minimum instead, replace OR with AND. This is MIN-private, as in the OR protocol above.

When the domain is large (e.g., we want the MAX of 64-bit packet counters, in a networking application), we can get a c-approximation of the MIN and MAX using a similar idea: divide the range $\{0, \ldots, B-1\}$ into $b = \log_c B$ "bins" $[0, c), [c, c^2), \ldots, [c^{b-1}, B)$. Then, use the small-range MIN/MAX construction, over the b bins, to compute the approximate statistic. The output will be within a multiplicative factor of c of the true value. This construction is private with respect to the approximate MIN/MAX function.

Frequency count. Here, every client has a value x in a small set of data values $\mathcal{D} = \{0, \ldots, B-1\}$. The goal is to output a B-element vector $v = (v^{(1)}, \ldots, v^{(B)})$, where $v^{(i)}$ is the number of clients that hold the value i, for every $0 \leq i < B$.

Let $\mathbb{F}$ be a field of size at least n. The Encode algorithm encodes a value $x \in \mathcal{D}$ as a length-B vector $(\beta_0, \ldots, \beta_{B-1}) \in \mathbb{F}^B$ where $\beta_i = 1$ if $x = i$ and $\beta_i = 0$ otherwise. The Valid algorithm checks that each β value is in the set $\{0, 1\}$ and that the sum of the βs is exactly one. The Dec algorithm does nothing: the final output is a length-B vector, whose ith component gives the number of clients who took on value i. Again, this AFE is private with respect to the function being computed.

The output of this AFE yields enough information to compute other useful functions (e.g., quantiles) of the distribution of the clients' x values. When the domain $\mathcal{D}$ is large, this AFE is very inefficient.

Sets. We can compute the intersection or union of sets over a small universe of elements using the boolean AFE operations: represent a set of B items as its characteristic vector of booleans, and compute an AND for intersection and an OR for union. When the universe is large, the approximate AFEs of Section 4.5.5 are more efficient.

4.5.4 Machine learning

We can use Prio for training machine learning models on private client data. To do so, we exploit the observation of Karr et al. [179] that a system for computing private sums can also privately train linear models. In Prio, we extend their work by showing how to perform these tasks while maintaining robustness against malicious clients.

Suppose that every client holds a data point (x, y) where x and y are b-bit integers. We would like to train a model that takes x as input and outputs a real-valued prediction $\hat{y}_i = \mathcal{M}(x) \in \mathbb{R}$ of y. We might predict a person's blood pressure (y) from the number of steps they walk daily (x).

We wish to compute the least-squares linear fit $h(x) = c_0 + c_1 x$ over all of the client points. With n clients, the model coefficients c_0 and c_1 satisfy the linear relation:

$$\begin{pmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{pmatrix} \tag{4.1}$$

To compute this linear system in an AFE, every client encodes her private point (x, y) as a vector

$$(x, x^2, y, xy, \quad \beta_0, \ldots, \beta_{b-1}, \quad \gamma_0, \ldots, \gamma_{b-1}) \in \mathbb{F}^{2b+4},$$

where $(\beta_0, \ldots, \beta_{b-1})$ is the binary representation of x and $(\gamma_0, \ldots, \gamma_{b-1})$ is the binary representation of y. The validation algorithm checks that all the β and γ are in $\{0, 1\}$, and that all the arithmetic relations hold, analogously to the validation check for the integer summation AFE. Finally, the decoding algorithm takes as input the sum of the encoded vectors truncated to the first four components:

$$\sigma = \left(\sum_{i=1}^{n} x, \quad \sum_{i=1}^{n} x^2, \quad \sum_{i=1}^{n} y, \quad \sum_{i=1}^{n} xy \right),$$

from which the decoding algorithm computes the required real regression coefficients c_0 and c_1 using (4.1). This AFE is private with respect to the function that outputs the least-squares fit $h(x) = c_0 + c_1 x$, along with the mean and variance of the set $\{x_1, \ldots, x_n\}$.

When x and y are real numbers, we can embed the reals into a finite field $\mathbb{F}$ using a fixed-point representation, as long as we size the field large enough to avoid overflow.

The two-dimensional approach above generalizes directly to perform linear regression on d-dimensional feature vectors $\bar{x} = (x^{(1)}, \ldots, x^{(d)})$. The AFE yields a least-squares approximation of the form $h(\bar{x}) = c_0 + c_1 x^{(1)} + \cdots + c_d x^{(d)}$. The resulting AFE is private with respect to a function that reveals the least-square coefficients $(c_0, \ldots, c_d)$, along with the $d \times d$ covariance matrix $\sum_{i=1}^{n} \bar{x}_i \cdot (\bar{x}_i)^T$.

Evaluating an arbitrary ML model. We wish to measure how well a public regression model predicts a target y from a client-submitted feature vector x. In particular, if our model outputs a prediction $\hat{y} = \mathcal{M}(x)$, we would like to measure how good of an approximation $\hat{y}$ is of y. The R^2 coefficient is one statistic for capturing this information.

Karr et al. [179] observe that it is possible to reduce the problem of computing the R^2 coefficient of a public regression model to the problem of computing private sums. We can adopt a variant of this idea to use Prio to compute the R^2 coefficient in a way that leaks little beyond the coefficient itself.

The R^2-coefficient of the model for client inputs $\{x_1, \ldots, x_n\}$ is $R^2 = 1 - \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 / \mathrm{Var}(y_1, \ldots, y_n)$, where y_i is the true value associated with x_i, $\hat{y}_i = \mathcal{M}(x_i)$ is the predicted value of y_i, and $\mathrm{Var}(\cdot)$ denotes variance.

An AFE for computing the R^2 coefficient works as follows. On input (x, y), the Encode algorithm first computes the prediction $\hat{y} = \mathcal{M}(x)$ using the public model $\mathcal{M}$. The Encode algorithm then outputs the tuple $(y, y^2, (y - \hat{y})^2, x)$, embedded in a finite field large enough to avoid overflow.

Given the tuple (y, Y, Y^*, x) as input, the Valid algorithm ensures that $Y = y^2$ and $Y^* = (y - \mathcal{M}(x))^2$. When the model $\mathcal{M}$ is a linear regression model, algorithm Valid can be represented as an arithmetic circuit that requires only two multiplications. If needed, we can augment this with a check that the x values are integers in the appropriate range using a range check, as in prior AFEs. Finally, given the sum of encodings restricted to the first three components, the Dec algorithm has the information it needs to compute the R^2 coefficient.

This AFE is private with respect to a function that outputs the R^2 coefficient, along with the expectation and variance of $\{y_1, \ldots, y_n\}$.

4.5.5 Approximate counts

The frequency count AFE, presented in Section 4.5.3, works well when the client value x lies in a small set of possible data values $\mathcal{D}$. This AFE requires communication linear in the size of $\mathcal{D}$. When the set $\mathcal{D}$ is large, a more efficient solution is to use a randomized counting data structure, such as a count-min sketch [91].

Melis et al. [207] demonstrated how to combine a count-min sketch with a secret-sharing scheme to efficiently compute counts over private data. We can make their approach robust to malicious clients by implementing a count-min sketch AFE in Prio. To do so, we use $\ln(1/\delta)$ instances of the basic frequency count AFE, each for a set of size e/ϵ, for some constants ϵ and δ, and where $e \approx 2.718$. With n client inputs, the count-min sketch yields counts that are at most an additive ϵn overestimate of the true values, except with probability $e^{-\delta}$.

Crucially, the Valid algorithm for this composed construction requires a relatively small number of multiplication gates—a few hundreds, for realistic choices of ϵ and δ—so the servers can check the correctness of the encodings efficiently.

This AFE leaks the contents of a count-min sketch data structure into which all of the clients' values $(x_1, \ldots, x_n)$ have been inserted.

One additional subtlety is that malicious clients may choose their inputs in such a way that depends on the randomness used in sketching data structure. So, we need to take care that such dependencies cannot break robustness. When using the count-min sketch as we do, it follows that if there are m malicious clients, then the data structure gives an approximate count that is an overestimate by at most an $\epsilon n + m$ additive term. So if the faction of malicious clients m/n is less than ϵ, we the structure provides an $\epsilon' < 2\epsilon$ approximation of the true counts, except with probability $e^{-\delta}$.

Share compression. The output of the count-min sketch AFE encoding routine is essentially a very sparse matrix of dimension $\ln(1/\delta) \times (e/\epsilon)$. The matrix is all zeros, except for a single "1" in each row. If the Prio client uses a conventional secret-sharing scheme to split this encoded matrix into s shares—one per server—the size of each share would be as large as the matrix itself, even though the plaintext matrix contents are highly compressible.

A more efficient way to split the matrix into shares would be to use a function secret-sharing scheme [62, 63, 138]. Applying a function secret sharing scheme to each row of the encoded matrix would allow the size of each share to grow as the square-root of the matrix width (instead of linearly). When using Prio with only two servers, there are very efficient function secret-sharing constructions that would allow the shares to have length logarithmic in the width of the matrix [63]. We leave further exploration of this technique to future work.

Most popular. Another common task is to return the most popular string in a data set, such as the most popular homepage amongst a set of Web clients. When the universe of strings is small, it is possible to find the most popular string using the frequency-counting AFE. When the universe is large (e.g., the set of all URLs), this method is not useful, since recovering the most popular string would require querying the structure for the count of every possible string. Instead, we use a simplified version of a data structure of Bassily and Smith [21].

When there is a very popular string—one that more than $n/2$ clients hold, we can construct a very efficient AFE for collecting it. Let $\mathbb{F}$ be a field of size at least n. The $\mathsf{Encode}(x)$ algorithm represents its input x as a b-bit string $x = (x_0, x_1, x_2, \ldots, x_{b-1}) \in \{0,1\}^b$, and outputs a vector of b field elements $(\beta_0, \ldots, \beta_{b-1}) \in \mathbb{F}^b$, where $\beta_i = x_i$ for all i. The Valid algorithm uses b multiplication gates to check that each value β_i is really a 0/1 value in $\mathbb{F}$, as in the summation AFE.

The Dec algorithm gets as input the sum of n such encodings $\sigma = \sum_{i=1}^{n} \mathsf{Encode}(x_i) = (e_0, \ldots, e_{b-1}) \in \mathbb{F}^b$. The Dec algorithm rounds each value e_i either down to zero or up to n (whichever is closer) and then normalizes the rounded number by n to get a b-bit binary string $\sigma \in \{0,1\}^b$, which it outputs. As long as there is a string σ^* with popularity greater than 50%, this AFE returns it. To see why, consider the first bit of σ. If $\sigma^*[0] = 0$, then the sum $e_0 < n/2$ and Dec outputs "0." If $\sigma^*[0] = 1$, then the sum $e_0 > n/2$ and Dec outputs "1." Correctness for longer strings follows

This AFE leaks quite a bit of information about the given data. Given σ, one learns the number of data values that have their ith bit set to 1, for every $0 \leq i < b$. In fact, the AFE

is private relative to a function that outputs these b values, which shows that nothing else is leaked by σ.

With a significantly more complicated construction, we can adapt a similar idea to collect strings that a constant fraction c of clients hold, for $c \leq 1/2$. The idea is to have the servers drop client-submitted strings at random into different "buckets," such that at least one bucket has a very popular string with high probability [21].

4.6 Prio protocol and proof sketch

We now assemble the pieces of the full Prio protocol in Fig. 4.2 and then discuss its security.

Security. We briefly sketch the security argument for the complete protocol.

First, the robustness property (Definition 4.2.6) follows from the soundness of the zero-knowledge proof on secret-shared data: a set of honest servers will correctly identify and reject any client submissions that do not represent proper AFE encodings.

Next, we argue f-privacy (Definition 4.2.1). Define the function

$$g(x_1, \ldots, x_{n-m}) = \sum_{i=1}^{n-m} \mathsf{Trunc}_{k'}\big(\mathsf{Encode}(x_i)\big).$$

We claim that, as long as:

- at least one server executes the protocol correctly,
- the AFE construction is private with respect to f, in the sense of Definition 4.5.3, and
- the proof on secret-shared data satisfies the zero-knowledge property (Section 4.4.1),

the only information that leaks to the adversary is the value of the function f on the private values of the honest clients included in the final aggregate.

To show this, it suffices to construct a simulator S that takes as input $\sigma = g(x_1, \ldots, x_{n-m})$ and outputs a transcript of the protocol execution that is indistinguishable from a real transcript. Recall that the AFE simulator takes $f(x_1, \ldots, x_{n-m})$ as input and simulates σ. Composing the simulator S with the AFE simulator yields a simulator for the entire protocol, as required by Definition 4.2.1.

On input σ, the simulator S executes these steps:

- To simulate the submission of each honest client, the simulator invokes the proof system's simulator as a subroutine.

The final Prio protocol. We first review the Prio protocol from Section 4.5. Let there be m malicious clients whose values are included in the final aggregate. We assume that every honest client i, for $i \in \{1, \ldots, n-m\}$, holds a private value x_i that lies in some set of data items $\mathcal{D}$. We want to compute an aggregation function $f : \mathcal{D}^{n-m} \to \mathcal{A}$ on these private values using an AFE consisting of $(\mathsf{Encode}, \mathsf{Valid}, \mathsf{Dec})$ with parameters k and k'. The AFE encoding algorithm Encode maps $\mathcal{D}$ to $\mathbb{F}^k$, for some field $\mathbb{F}$ and an arity k. When decoding, the AFE truncates the encoded vectors in $\mathbb{F}^k$ to their first k' components.

The Prio protocol proceeds in four steps:

1. **Upload.** Each client i computes $y_i \leftarrow \mathsf{Encode}(x_i)$ and splits its encoded value into s shares, one per server. To do so, the client picks random values $[\![y_i]\!]_1, \ldots, [\![y_i]\!]_s \in \mathbb{F}^k$, subject to the constraint: $y_i = [\![y_i]\!]_1 + \cdots + [\![y_i]\!]_s \in \mathbb{F}^k$. The client then sends, over an encrypted and authenticated channel, one share of its submission to each server, along with a share of a one-round zero-knowledge proof on secret-shared data (Section 4.4) asserting that $\mathsf{Valid}(y_i) = 1$.
2. **Validate.** Upon receiving the ith client submission, the servers verify the client-provided proof to jointly confirm that $\mathsf{Valid}(y_i) = 1$ (i.e., that client's submission is well-formed). If this check fails, the servers reject the submission.
3. **Aggregate.** Each server j holds an accumulator value $A_j \in \mathbb{F}^{k'}$, initialized to zero, where $0 < k' \leq k$. Upon receiving a share of a client encoding $[\![y_i]\!]_j \in \mathbb{F}^k$, Server j truncates $[\![y_i]\!]_j$ to its first k' components, and adds this share to its accumulator:

   $$A_j \leftarrow A_j + \mathsf{Trunc}_{k'}([\![y_i]\!]_j) \in \mathbb{F}^{k'}.$$

   Recall that $\mathsf{Trunc}_{k'}(v)$ denotes truncating the vector $v \in \mathbb{F}^k$ to its first k' components.
4. **Publish.** Once the servers have received a share from each client, they publish their accumulator values. The sum of the accumulator values $\sigma = \sum_{j=1}^{s} A_j \in \mathbb{F}^{k'}$ yields the sum $\sum_{i=1}^{n} \mathsf{Trunc}_{k'}(y_i)$ of the clients' private encoded values. The servers output $\mathsf{Dec}(\sigma)$.

Figure 4.2: The final Prio protocol.

- To simulate values produced by adversarial clients, the simulator can query the adversary (presented to the simulator as an oracle) on the honest parties' values generated so far.
- The simulator must produce a simulation of the values produced by the honest servers in the last step of the protocol.

  Let σ be the sum of the honest clients' encodings. For the simulation to be accurate, the honest servers must publish values A_j such that the honest servers' accumulators sum to (1) all of the shares of encodings given to the honest servers by adversarial clients plus (2) σ minus (3) the shares of honest clients' simulated encodings given to adversarial servers.

  Let $[\![y_i]\!]_j$ be the jth share of the encoding sent by client i. Let $\mathsf{Serv}_{\mathcal{A}}$ be the set of indices of the adversarial servers and $\mathsf{Client}_{\mathcal{A}}$ be the set of indices of the adversarial clients. Let Serv_H and Client_H be the set of indices of the honest servers and clients, respectively. Then the accumulators A_j must satisfy the relation:

$$\sum_{j \in \mathsf{Serv}_H} A_j = \sigma + \left(\sum_{j \in \mathsf{Serv}_H} \sum_{i \in \mathsf{Client}_{\mathcal{A}}} [\![y_i]\!]_j \right) - \left(\sum_{j \in \mathsf{Serv}_{\mathcal{A}}} \sum_{i \in \mathsf{Client}_H} [\![y_i]\!]_j \right).$$

  The simulator can pick random values for the honest servers' A_js subject to this constraint, since the simulator knows σ, it knows the values that the honest clients' sent to the adversarial servers, and it knows the values that the adversarial clients sent to the honest servers.

To argue that the simulator S correctly simulates a run of the real protocol:

- The existence of the proofs' simulator implies that everything the adversarial servers see in the proof-verification step, when interacting with an honest client, is independent of the honest client's private value x_i, provided that $\mathsf{Valid}(x_i) = 1$. This property holds even if the malicious servers deviate from the protocol in a way that causes an honest client's submission to be rejected by the honest servers.
- The simulation of the aggregation step of the protocol is perfect. In the real protocol, since the adversary sees only $s - 1$ shares of each client submission, the values A_j are just random values subject to the constraint above.

Finally, anonymity (Definition 4.2.2) follows by Claim 4.2.4 whenever the function f is symmetric. Otherwise, anonymity is impossible, by Claim 4.2.5.

4.7 Additional optimizations

We discuss a number of additional optimizations to the protocol that are useful in practice.

Optimization: PRG secret sharing. The Prio protocol uses additive secret sharing to split the clients' private data into shares. The naïve way to split a value $x \in \mathbb{F}^k$ into s shares is to choose $[\![x]\!]_1, \ldots, [\![x]\!]_{s-1} \in \mathbb{F}^k$ at random and then set $[\![x]\!]_s = x - \sum_{i=1}^{s-1} [\![x]\!]_i \in \mathbb{F}^k$. A standard bandwidth-saving optimization is to generate the first $s-1$ shares using a pseudo-random generator (PRG) $G : \mathcal{S} \to \mathbb{F}^k$, such as AES in counter mode [187, 236]. To do so, pick $s-1$ random PRG seed $\sigma_1, \ldots, \sigma_{s-1} \in \mathcal{S}$, and define the first $s-1$ shares as $G(\sigma_1), G(\sigma_2), \ldots, G(\sigma_{s-1})$. Rather than representing the first $s-1$ shares as vectors in $\mathbb{F}^k$, we can now represent each of the first $s-1$ shares using a single PRG seed. (The last share will still be k field elements in length.) This optimization reduces the total size of the shares from sk field elements down to $k + O(1)$. For $s = 5$ servers, this $5\times$ bandwidth savings is significant.

Optimization: Verification without interpolation. In the course of verifying our zero-knowledge proofs on secret-shared data, constructed using our fully linear PCPs (Theorem 2.3.3), each server needs to produce the query vector for the fully linear PCP construction and then compute the inner product of these queries with its share of the input and proof. Concretely, to produce these query vectors, each server j needs to interpolate two large polynomials $[\![f]\!]_j$ and $[\![g]\!]_j$. Then, each server must evaluate these polynomials $[\![f]\!]_j$ and $[\![g]\!]_j$ at a randomly chosen point $r \in \mathbb{F}$.

The degree of these polynomials is M, where M is the number of multiplication gates in the Valid circuit. If the servers used straightforward polynomial interpolation and evaluation to verify the proofs on secret-shared data, the servers would need to perform $\Theta(M \log M)$ multiplications to process a single client submission, even using optimized FFT methods. When the Valid circuit is complex (i.e., $M \approx 2^{16}$ or more), this $\Theta(M \log M)$ cost will be substantial.

Let us imagine for a minute that we could fix *in advance* the random point r that the servers use to execute the polynomial identity test. In this case, each server can perform interpolation and evaluation of any polynomial P in one step using only $O(M)$ field multiplications per server, instead of $\Theta(M \log M)$. To do so, each server precomputes constants $(c_0, \ldots, c_M) \in \mathbb{F}^{M+1}$. These constants depend on the x-coordinates of the points being interpolated (which

are always fixed in our application) and on the point r (which for now we assume is fixed). Then, given points $\{(t, y_t)\}_{t=0}^{M}$ on a polynomial P, the servers can evaluate P at r using a fast inner-product computation: $P(x) = \sum_{t=0}^{M} c_t y_t \in \mathbb{F}$. Standard Lagrangian interpolation produces these c_is as intermediate values [13].

Our observation is that the servers *can* fix the "random" point r at which they evaluate the polynomials $[\![f]\!]_j$ and $[\![g]\!]_j$ as long as: (1) the clients never learn r, and (2) the servers sample a new random point r periodically. The randomness of the value r only affects soundness. Since we require soundness to hold only if all Prio servers are honest, we may assume that the servers will never reveal the value r to the clients.

A malicious client may try to learn something over time about the servers' secret value r by sending a batch of well-formed and malformed submissions and seeing which submissions the servers do or do not accept. After making Q such queries, the client's probability of cheating the servers is at most $(2M + 1)Q/|\mathbb{F}|$. By sampling a new point after every $Q \approx 2^{10}$ client uploads, the servers can amortize the cost of doing the interpolation precomputation over Q client uploads, while keeping the failure probability bounded above by $(2M+1)Q/|\mathbb{F}|$, which they might take to be 2^{-60} or less.

In Prio, we apply this optimization to combine the interpolation of $[\![f]\!]_j$ and $[\![g]\!]_j$ with the evaluation of these polynomials at the point r.

During the proof verification process, each server must also evaluate the client-provided polynomial $[\![h]\!]_j$ at each point $t \in \{0, \ldots, M\}$. To eliminate this cost, we have the client send the polynomial $[\![h]\!]_j$ to each server j in point-value form. That is, instead of sending each server shares of the coefficients of h, the client sends each server shares of evaluations of h. In particular, the client evaluates $[\![h]\!]_j$ at all of the points $t \in \{0, \ldots, 2M\}$ and sends the evaluations $[\![h]\!]_j(0), [\![h]\!]_j(1), [\![h]\!]_j(2), \ldots, [\![h]\!]_j(2M)$ to the server. Now, each server j already has the evaluations of $[\![h]\!]_j$ at all of the points it needs to complete the proof verification. To check the proof, each server must interpolate $[\![h]\!]_j$ and evaluate $[\![h]\!]_j$ at the point r. We accomplish this using the same fast interpolation-and-evaluation trick described above for $[\![f]\!]_j$ and $[\![g]\!]_j$

Circuit optimization In many cases, the servers hold a batch of verification circuits $\mathsf{Valid}_1, \ldots, \mathsf{Valid}_B$ and want to check whether the client's submission passes all B checks. To do so, we apply another standard optimization and have the Valid circuits return zero (instead of one) on success. If W_ℓ is the value on the last output wire of the circuit Valid_ℓ, we have the servers choose random values $(r_1, \ldots, r_B) \in \mathbb{F}^B$ and publish the sum $\sum_{\ell=1}^{B} r_\ell W_\ell$

		Workstation		Phone	
	Field size:	87-bit	265-bit	87-bit	265-bit
Mul. in field (μs)		1.013	1.485	11.218	14.930
Prio client time (s)	$k = 10^1$	0.003	0.004	0.017	0.024
	$k = 10^2$	0.024	0.036	0.112	0.170
	$k = 10^3$	0.221	0.344	1.059	2.165

Table 4.2: Time in seconds for a client to generate a Prio submission of k four-bit integers to be summed at the servers. Averaged over eight runs.

in the last step of the protocol. If any $W_\ell \neq 0$, then this sum will be non-zero with high probability and the servers will reject the client's submission.

4.8 Evaluation

In this section, we demonstrate that Prio's theoretical contributions translate into practical performance gains. We have implemented a Prio prototype in 5,700 lines of Go and 620 lines of C (for FFT-based polynomial operations, built on the FLINT library [127]). Unless noted otherwise, our evaluations use an FFT-friendly 87-bit field. Our servers communicate with each other using Go's TLS implementation. Clients encrypt and sign their messages to servers using NaCl's "box" primitive, which obviates the need for client-to-server TLS connections. Our code is available online at https://crypto.stanford.edu/prio/.

We evaluate the primary variant of Prio, which uses our new proofs on secret-shared data (Section 4.4.1) and also the variant in which the servers keep the Valid predicate private ("Prio-MPC," Section 4.4.2). Our implementation includes three optimizations described in Section 4.7. The first uses a pseudo-random generator (e.g., AES in counter mode) to reduce the client-to-server data transfer by a factor of roughly s in an s-server deployment. The second optimization allows the servers to verify the proofs without needing to perform expensive polynomial interpolations (see Section 4.7). The third optimization gives an efficient way for the servers to compute the logical-AND of multiple arithmetic circuits to check that multiple Valid predicates hold simultaneously.

We compare Prio against a private aggregation scheme that uses non-interactive zero-knowledge proofs (NIZKs) to provide robustness. This protocol is similar to the "cryptographically verifiable" interactive protocol of Kursawe et al. [188] and has roughly the same

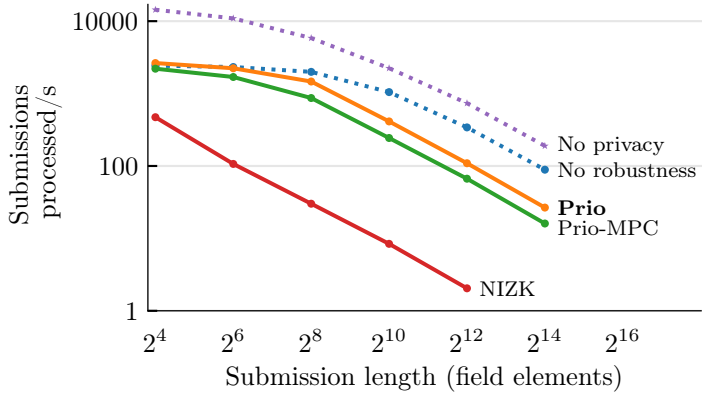


Figure 4.3: Prio's proofs on secret-shared data provide the robustness guarantees of standard zero-knowledge proofs at 20-50× less cost.

cost, in terms of exponentiations per client request, as the "distributed decryption" variant of PrivEx [119]. We implement the discrete-log-based NIZK scheme using a Go wrapper of OpenSSL's NIST P256 code [109]. We do not compare Prio against systems, such as ANONIZE [165] and PrivStats [231], that rely on an external anonymizing proxy to protect against a network adversary. (We discuss this related work in Section 4.10.)

4.8.1 Microbenchmarks

Table 4.2 presents the time required for a Prio client to encode a data submission on a workstation (2.4 GHz Intel Xeon E5620) and mobile phone (Samsung Galaxy SIII, 1.4 GHz Cortex A9). For a submission of 100 integers, the client time is roughly 0.03 seconds on a workstation, and just over 0.1 seconds on a mobile phone.

To investigate the load that Prio places on the servers, we configured five Amazon EC2 servers (eight-core c3.2xlarge machines, Intel Xeon E5-2680 CPUs) in five Amazon data centers (N. Va., N. Ca., Oregon, Ireland, and Frankfurt) and had them run the Prio protocols. An additional three c3.2xlarge machines in the N. Va. data center simulated a large number of Prio clients. To maximize the load on the servers, we had each client send a stream of pre-generated Prio data packets to the servers over a single TCP connection. There is no need to use TLS on the client-to-server Prio connection because Prio packets are encrypted and authenticated at the application layer and can be replay-protected at the servers.

Figure 4.3 gives the throughput of this cluster in which each client submits a vector of

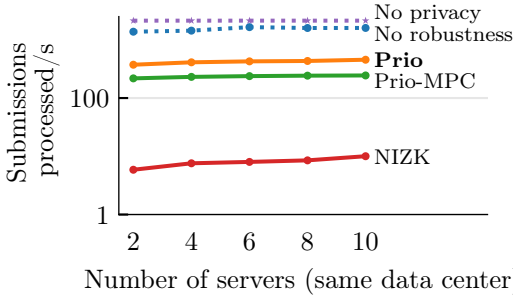


Figure 4.4: Prio is insensitive to the number of aggregation servers.

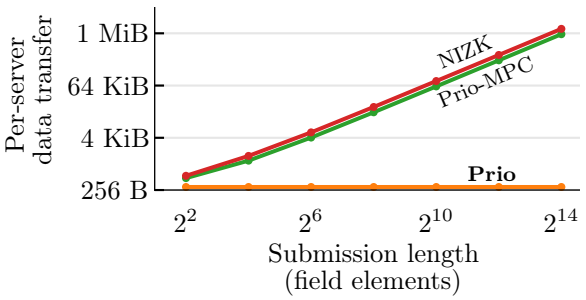


Figure 4.5: Prio's use of proofs on secret-shared data (§4.4) reduces bandwidth consumption.

zero/one integers and the servers sum these vectors. The "No privacy" line on the chart gives the throughput for a dummy scheme in which a single server accepts encrypted client data submissions directly from the clients with no privacy protection whatsoever. The "No robustness" line on the chart gives the throughput for a cluster of five servers that use a secret-sharing-based private aggregation scheme (*à la* Section 4.3) with no robustness protection. The five-server "No robustness" scheme is slower than the single-server "No privacy" scheme because of the cost of coordinating the processing of submissions amongst the five servers. The throughput of Prio is within a factor of 5× of the no-privacy scheme for many submission sizes, and Prio outperforms the NIZK-based scheme by more than an order of magnitude.

Figure 4.4 shows how the throughput of a Prio cluster changes as the number of servers increases, when the system is collecting the sum of 1,024 one-bit client-submitted integers, as in an anonymous survey application. For this experiment, we locate all of the servers in the same data center, so that the latency and bandwidth between each pair of servers is roughly constant. With more servers, an adversary has to compromise a larger number of machines to violate Prio's privacy guarantees.

Adding more servers barely affects the system's throughput. The reason is that we are able to load-balance the bulk of the work of checking client submissions across all of the servers. (This optimization is only possible because we require robustness to hold only if all servers are honest.) We assign a single Prio server to be the "leader" that coordinates the checking of each client data submission. In processing a single submission in an s-server cluster, the leader transmits s times more bits than a non-leader, but as the number of servers increases, each server is a leader for a smaller share of incoming submissions. The NIZK-based

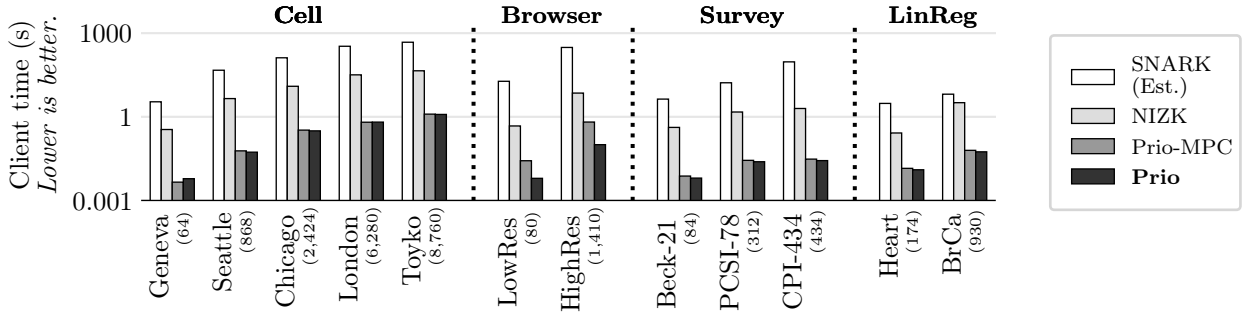


Figure 4.6: Client encoding time for different application domains when using Prio, a discrete-log-based non-interactive zero-knowledge system (NIZK), or a SNARK-like system (estimated). Averaged over eight runs. We list the number of multiplication gates in the Valid circuit in parentheses.

scheme also scales well: as the number of servers increases, the heavy computational load of checking the NIZKs is distributed over more machines.

Figure 4.5 shows the number of bytes each non-leader Prio server needs to transmit to check the validity of a single client submission for the two Prio variants, and for the NIZK scheme. The benefit of Prio is evident: the Prio servers transmit a constant number of bits per submission—*independent* of the size of the submission or complexity of the Valid routine. As the submitted vectors grow, Prio yields a 4,000-fold bandwidth saving over NIZKs, in terms of server data transfer.

4.8.2 Application scenarios

To demonstrate that Prio's data types are expressive enough to collect real-world aggregates, we have configured Prio for a few potential application domains.

Cell signal strength. A collection of Prio servers can collect the average mobile signal strength in each grid cell in a city without leaking the user's location history to the aggregator. We divide the geographic area into a km^2 grid—the number of grid cells depends on the city's size—and we encode the signal strength at the user's present location as a four-bit integer. (If each client only submits signal-strength data for a few grid cells in each protocol run, extra optimizations can reduce the client-to-server data transfer. See "Share compression" in Section 4.5.5.)

Browser statistics. The Chromium browser uses the RAPPOR system to gather private information about its users [87, 121]. We implement a Prio instance for gathering a subset

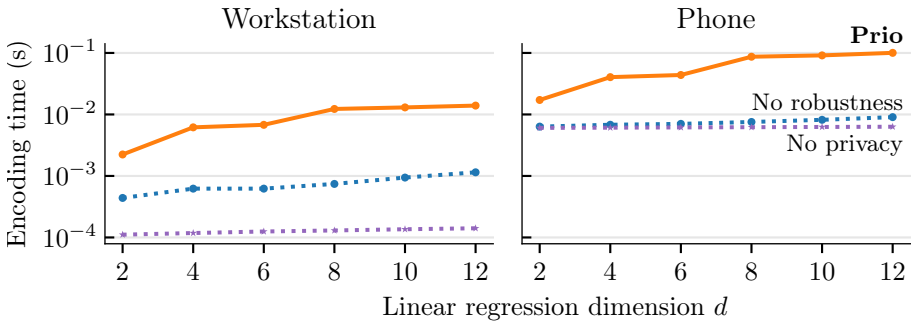


Figure 4.7: Time for a client to encode a submission consisting of d-dimensional training example of 14-bit values for computing a private least-squares regression.

of these statistics: average CPU and memory usage, along with the frequency counts of 16 URL roots. For collecting the approximate counts, we use the count-min sketch structure, described in Section 4.5.5. We experiment with both low- and high-resolution parameters ($\delta = 2^{-10}$, $\epsilon = 1/10$; $\delta = 2^{-20}$, $\epsilon = 1/100$).

Health data modeling. We implement the AFE for training a regression model on private client data. We use the features from a preexisting heart disease data set (13 features of varying types: age, sex, cholesterol level, etc.) [171] and a breast cancer diagnosis data set (30 real-valued features using 14-bit fixed-point numbers) [273].

Anonymous surveys. We configure Prio to compute aggregates responses to sensitive surveys: we use the Beck Depression Inventory (21 questions on a 1-4 scale) [7], the Parent-Child Relationship Inventory (78 questions on a 1-4 scale) [135], and the California Psychological Inventory (434 boolean questions) [98].

Comparison to alternatives. In Figure 4.6, we compare the computational cost Prio places on the client to the costs of other schemes for protecting robustness against misbehaving clients, when we configure the system for the aforementioned applications. The fact that a Prio client need only perform a single public-key encryption means that it dramatically outperforms schemes based on public-key cryptography. If the Valid circuit has M multiplication gates, producing a discrete-log-based NIZK requires the client to perform $2M$ exponentiations (or elliptic-curve point multiplications). In contrast, Prio requires $O(M \log M)$ multiplications in a relatively small field, which is much cheaper for practical values of M.

In Figure 4.6, we give conservative estimates of the time required to generate a zkSNARK proof, based on timings of libsnark's [41] implementation of the Pinocchio system [225]

	No privacy	No robustness		**Prio**		
d	Rate	Rate	Priv. cost	Rate	Robust. cost	Tot. cost
2	14,688	2,687	5.5×	2,608	1.0×	5.6×
4	15,426	2,569	6.0×	2,165	1.2×	7.1×
6	14,773	2,600	5.7×	2,048	1.3×	7.2×
8	15,975	2,564	6.2×	1,606	1.6×	9.5×
10	15,589	2,639	5.9×	1,430	1.8×	10.9×
12	15,189	2,547	6.0×	1,312	1.9×	11.6×

Table 4.3: The throughput, in client requests per second, of a global five-server cluster running a private d-dim. regression. We compare a scheme with no privacy, with privacy but no robustness, and Prio (with both).

at the 128-bit security level. These proofs have the benefit of being very short: 288 bytes, irrespective of the complexity of the circuit. To realize the benefit of these succinct proofs, the statement being proved must also be concise since the verifier's running time grows with the statement size. To achieve this conciseness in the Prio setting would require computing sn hashes "inside the SNARK," with s servers and submissions of length n.

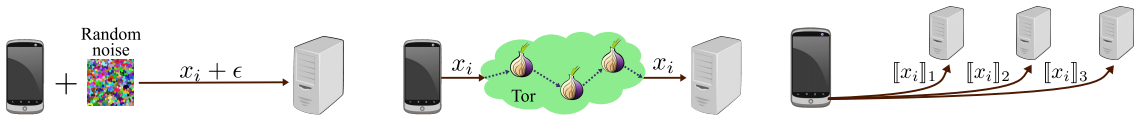
We optimistically estimate that each hash computation requires only 300 multiplication gates, using a subset-sum hash function [4, 40, 145, 167], and we ignore the cost of computing the Valid circuit in the SNARK. We then use the timings from the libsnark paper to arrive at the cost estimates. Each SNARK multiplication gate requires the client to compute a number of exponentiations, so the cost to the client is large, though the proof is admirably short.

4.8.3 Machine learning

Finally, we perform an end-to-end evaluation of Prio when the system is configured to train a d-dimensional least-squares regression model on private client-submitted data, in which each training example consists of a vector of 14-bit integers. These integers are large enough to represent vital health information, for example.

In Figure 4.7, we show the client encoding cost for Prio, along with the no-privacy and no-robustness schemes described in Section 4.8.1. The cost of Prio's privacy and robustness guarantees amounts to roughly a 50× slowdown at the client over the no-privacy scheme due to the overhead of the proof generation. Even so, the absolute cost of Prio to the client is small—on the order of one tenth of a second.

Table 4.3 gives the rate at which the globally distributed five-server cluster described



(a) RAPPOR [121] provides differential privacy [115] (not information-theoretic privacy) by adding random noise to client submissions.

(b) ANONIZE [165] and PrivStats [231] rely on an anonymizing proxy, such as Tor [112], to protect privacy against network eavesdroppers.

(c) Prio and other schemes using secret sharing [72, 107, 119, 173, 188, 207] offer ideal anonymity provided that the servers do not collude.

Figure 4.8: Comparison of techniques for anonymizing client data in private aggregation systems.

in Section 4.8.1 can process client submissions with and without privacy and robustness. The server-side cost of Prio is modest: only a 1-2× slowdown over the no-robustness scheme, and only a 5-15× slowdown over a scheme with *no privacy at all*. In contrast, the cost of robustness for the state-of-the-art NIZK schemes, per Figure 4.3, is closer to 100-200×.

4.9 Discussion

Deployment scenarios. Prio ensures client privacy as long as at least one server behaves honestly. We now discuss a number of deployment scenarios in which this assumption aligns with real-world incentives.

Tolerance to compromise. Prio lets an organization compute aggregate data about its clients without ever storing client data in a single vulnerable location. The organization could run all s Prio servers itself, which would ensures data privacy against an attacker who compromises up to $s - 1$ servers.

App store. A mobile application platform (e.g., Apple's App Store or Google's Play) can run one Prio server, and the developer of a mobile app can run the second Prio server. This allows the app developer to collect aggregate user data without having to bear the risks of holding these data in the clear.

Shared data. A group of s organizations could use Prio to compute an aggregate over the union of their customers' datasets, without learning each other's private client data.

Private compute services. A large enterprise can contract with an external auditor or a non-profit (e.g., the Electronic Frontier Foundation) to jointly compute aggregate statistics over sensitive customer data using Prio.

Jurisdictional diversity. A multinational organization can spread its Prio servers across different countries. If law enforcement agents seize the Prio servers in one country, they cannot deanonymize the organization's Prio users.

Common attacks. Two general attacks apply to *all systems*, like Prio, that produce exact (un-noised) outputs while protecting privacy against a network adversary. The first attack is a *selective denial-of-service attack*. In this attack, the network adversary prevents all honest clients except one from being able to contact the Prio servers [243]. In this case, the protocol output is $f(x_{\text{honest}}, x_{\text{evil}_1}, \ldots, x_{\text{evil}_n})$. Since the adversary knows the x_{evil} values, the adversary could infer part or all of the one honest client's private value x_{honest}.

In Prio, we deploy the standard defense against this attack, which is to have the servers wait to publish the aggregate statistic $f(x_1, \ldots, x_n)$ until they are confident that the aggregate includes values from many honest clients. The best means to accomplish this will depend on the deployment setting.

One way is to have the servers keep a list of public keys of registered clients (e.g., the students enrolled at a university). Prio clients sign their submissions with the signing key corresponding to their registered public key and the servers wait to publish their accumulator values until a threshold number of registered clients have submitted valid messages. Standard defenses [6, 263, 279, 280] against Sybil attacks [113] would apply here.

The second attack is an *intersection attack* [45, 108, 182, 275]. In this attack, the adversary observes the output $f(x_1, \ldots, x_n)$ of a run of the Prio protocol with n honest clients. The adversary then forces the nth honest client offline and observes a subsequent protocol run, in which the servers compute $f(x'_1, \ldots, x'_{n-1})$. If the clients' values are constant over time ($x_i = x'_i$), then the adversary learns the difference $f(x_1, \ldots, x_n) - f(x_1, \ldots, x_{n-1})$, which could reveal client n's private value x_n (e.g., if f computes SUM).

One way for the servers to defend against the attack is to add differential privacy noise to the results before publishing them [115]. Using existing techniques, the servers can add this noise in a distributed fashion to ensure that as long as at least one server is honest, no server sees the un-noised aggregate [116]. The definition of differential privacy ensures that computed statistics are distributed approximately the same whether or not the aggregate includes a particular client's data. This same approach is also used in a system by Melis, Danezis, and De Cristofaro [207], which we discuss in Section 4.10.

Robustness against malicious servers. Prio only provides robustness when all servers

are honest. Providing robustness in the face of faulty servers is obviously desirable, but we are not convinced that it is worth the security and performance costs. Briefly, providing robustness necessarily weakens the privacy guarantees that the system provides: if the system protects *robustness* in the presence of t faulty servers, then the system can protect *privacy* only against a coalition of at most $s - t - 1$ malicious servers. See Remark 4.2.7 for further discussion.

4.10 Related Work

Private data-collection systems [72, 107, 114, 119, 173, 188, 207] that use secret-sharing based methods to compute sums over private user data typically (a) provide no robustness guarantees in the face of malicious clients, (b) use expensive NIZKs to prevent client misbehavior, or (c) fail to defend privacy against actively malicious servers [80].

Other data-collection systems have clients send their private data to an aggregator through a general-purpose anonymizing network, such as a mix-net [64, 78, 104, 192] or a DC-net [77, 94, 95, 96, 252]. These anonymity systems provide strong privacy properties, but require expensive "verifiable mixing" techniques [23, 220], or require work at the servers that is *quadratic* in the number of client messages sent through the system [94, 274].

PrivStats [231] and ANONIZE [165] outsource to Tor [112] (or another low-latency anonymity system [129, 198, 235]) the work of protecting privacy against a network adversary (Figure 4.8). Prio protects against an adversary that can see and control the entire network, while Tor-based schemes succumb to traffic-analysis attacks [213].

In data-collection systems based on differential privacy [115], the client adds structured noise to its private value before sending it to an aggregating server. The added noise gives the client "plausible deniability:" if the client sends a value x to the servers, x could be the client's true private value, or it could be an unrelated value generated from the noise. Dwork et al. [116], Shi et al. [249], and Bassily and Smith [21] study this technique in a distributed setting, and the RAPPOR system [121, 124], deployed in Chromium, has put this idea into practice. A variant of the same principle is to have a trusted proxy (as in SuLQ [49] and PDDP [81]) or a set of minimally trusted servers [207] add noise to already-collected data.

The downside of these systems is that (a) if the client adds little noise, then the system does not provide much privacy, or (b) if the client adds a lot of noise, then low-frequency events may be lost in the noise [121]. Using server-added noise [207] ameliorates these

problems.

In theory, secure multi-party computation (MPC) protocols [26, 33, 146, 200, 278] allow a set of servers, with some non-collusion assumptions, to privately compute *any* function over client-provided values. The generality of MPC comes with serious bandwidth and computational costs: evaluating the relatively simple AES circuit in an MPC requires the parties to perform many minutes or even hours of precomputation [100]. Computing a function f on millions of client inputs, as our five-server Prio deployment can do in tens of minutes, could potentially take an astronomical amount of time in a full MPC. That said, there have been great advances in practical general-purpose MPC protocols of late [28, 30, 52, 101, 103, 159, 199, 204, 212, 228]. General-purpose MPC may yet become practical for computing certain aggregation functions that Prio cannot (e.g., exact MAX), and some special-case MPC protocols [12, 68, 221] are practical today for certain applications.

4.11 Conclusion and future work

Prio allows a set of servers to compute aggregate statistics over client-provided data while maintaining client privacy, defending against client misbehavior, and performing nearly as well as data-collection platforms that exhibit neither of these security properties. The core idea behind Prio is closely related to the techniques of verifiable computation [37, 92, 134, 149, 225, 265, 268, 272], but in reverse—the client proves to a set of servers that it computed a function correctly. One question for future work is whether it is possible to efficiently extend Prio to support combining client encodings using a more general function than summation, and what more powerful aggregation functions this would enable.

Chapter 5

Riposte: Anonymous messaging at million-user scale

5.1 Introduction

In a world of ubiquitous network surveillance [43, 132, 133, 142, 215], prospective whistleblowers face a daunting task. Consider, for example, a government employee who wants to anonymously leak evidence of waste, fraud, or incompetence to the public. The whistleblower could email an investigative reporter directly, but *post hoc* analysis of email server logs could easily reveal the tipster's identity. The whistleblower could contact a reporter via Tor [112] or another low-latency anonymizing proxy [129, 197, 211, 235], but this would leave the leaker vulnerable to traffic-analysis attacks [22, 213, 214]. The whistleblower could instead use an anonymous messaging system that protects against traffic analysis attacks [77, 141, 274], but these systems typically only support relatively small anonymity sets (tens of thousands of users, at most). Protecting whistleblowers in the digital age requires anonymous messaging systems that provide strong security guarantees, but that also scale to very large network sizes.

In this chapter, we present a new system that attempts to make traffic-analysis-resistant anonymous broadcast messaging practical at Internet scale. Our system, called Riposte, allows a large number of clients to anonymously post messages to a shared "bulletin board," maintained by a small set of minimally trusted servers. As few as two non-colluding servers are sufficient. Whistleblowers could use Riposte as a platform for anonymously publishing Tweet- or email-length messages and could combine it with standard public-key encryption

to build point-to-point private messaging channels.

While there is an extensive literature on anonymity systems [105, 118], Riposte offers a combination of security and scalability properties unachievable with prior designs. To the best of our knowledge, Riposte was the first anonymous messaging system that could simultaneously:

1. protect against traffic analysis attacks,
2. prevent malicious clients from anonymously executing denial-of-service attacks, and
3. scale to anonymity set sizes of *millions* of users, for certain latency-tolerant applications.

(See Section 5.7 for a discussion of subsequent related work.)

We achieve these three properties in Riposte by adapting three different techniques from the cryptography and privacy literature. First, we defeat traffic-analysis attacks and protect against malicious servers by using a protocol, inspired by client/server DC-nets [77, 274], in which every participating client sends a fixed-length secret-shared message to the system's servers in every time epoch. Second, we achieve efficient disruption resistance by applying our new zero-knowledge proofs on secret-shared data (Chapter 3). Third, we achieve scalability by leveraging a specific technique developed in the context of private information retrieval to minimize the number of bits each client must upload to each server in every time epoch. The tool we use is called a *distributed point function* [84, 138]. The novel synthesis of these techniques leads to a system that is efficient—in terms of bandwidth and computation—and arguably practical, even for large anonymity sets.

Our particular use of private information retrieval (PIR) protocols is unusual. PIR systems [86] allow a client to efficiently read a row from a database, maintained collectively at a set of servers, without revealing to the servers which row it is reading. Riposte achieves scalable anonymous messaging by running a private information retrieval protocol *in reverse*: with reverse PIR, a Riposte client can efficiently *write* into a database maintained at the set of servers without revealing to the servers which row it has written [223].

As we discuss later on, a large Riposte deployment could form the basis for an anonymous Twitter service. Users would "tweet" by using Riposte to anonymously write into a database containing all clients' tweets for a particular time period. In addition, by having read-only users submit "empty" writes to the system, the effective anonymity set can be much larger than the number of writers, with little impact on system performance.

Messaging in Riposte proceeds in regular *time epochs* that could be, for example, each

one hour long. To post a message, the client generates a *write request*, cryptographically splits it into many shares, and sends one share to each of the Riposte servers. A coalition of servers smaller than a certain threshold cannot learn anything about the client's message or write location given its subset of the shares.

The Riposte servers collect write requests until the end of the time epoch, at which time they publish the aggregation of the write requests they received during the epoch. From this information, anyone can recover the set of posts uploaded during the epoch, but the system reveals no information about who posted which message. The identity of the entire set of clients who posted during the interval is known, but no one can link a client to a post. Thus, each time epoch must be long enough to ensure that a large number of honest clients are able to participate in each epoch.

In this paper, we describe two Riposte variants, which offer slightly different security properties. The first variant is efficient enough to handle very large network sizes (millions of clients) but requires two non-colluding servers. The second variant is more computationally expensive, but it allows using any number of servers $s \geq 2$ and it provides security even when up to $s-1$ of the servers are malicious. Both variants maintain their security properties when network links are actively adversarial, when any number of the clients are actively malicious, and when the servers are actively malicious, subject to the non-collusion requirement above.

Unlike Tor [112] and other low-latency anonymity systems [141, 166, 197, 235], Riposte protects against active traffic analysis attacks by a global network adversary. Prior systems have offered traffic-analysis-resistance only at the cost of scalability:

- Mix-net-based systems [78] require expensive zero-knowledge proofs of correctness to provide privacy in the face of active attacks by malicious servers [3, 23, 130, 155, 220].
- DC-nets-based systems require clients to transfer data *linear* in the size of the anonymity set [77, 274] and rely on expensive zero-knowledge proofs to protect against malicious clients [96, 153].

We discuss these systems and other prior work in Section 5.7.

Experiments. To demonstrate the practicality of Riposte for anonymous broadcast messaging (i.e., anonymous whistleblowing or microblogging), we implemented and evaluated the two-server variant of the system. When the servers maintain a database table large enough to fit 65,536 160-byte Tweets, the system can process 240 client write requests per second. In Section 5.6.2, we discuss how to use a table of this size as the basis for very large anonymity

sets in read-heavy applications. When using a larger 377 MB database table (over 2.3 million 160-byte Tweets), a Riposte cluster can process 7.4 client write requests per second.

Writing into a 377 MB table requires each client to upload less than 2 MB of data to the servers. In contrast, a two-server DC-net-based system would require each client to upload more than 750 MB of data. More generally, to process a Riposte client request for a table of size L, clients and servers perform only $O(\sqrt{L})$ bytes of data transfer. Application of more recent cryptographic techniques [62, 63] could drop this cost to the asymptotically optimal $O(\log L)$ bytes. (Here, the big-O notation hides fixes polynomials in the security parameter.)

The servers' AES-NI encryption throughput limits the rate at which Riposte can process client requests at large table sizes. Thus, the system's capacity to handle client write request scales with the number of available CPU cores. A large Riposte deployment could shard the database table across μ machines to achieve a near-μ-fold speedup.

We tested the system with anonymity set sizes of up to 6,162,647 clients, with a read-heavy latency-tolerant microblogging workload. No prior system had *ever constructed* an anonymity set so large while defending against traffic analysis attacks. Prior DC-net-based systems scaled to 5,120 clients [274] and prior verifiable-shuffle-based systems scaled to 100,000 clients [23]. In contrast, Riposte scales to millions of clients for certain applications.

Contributions. This paper contributes:

- two new bandwidth-efficient and traffic-analysis-resistant anonymous messaging protocols, obtained by running private information retrieval protocols "in reverse" (Sections 5.3 and 5.4),
- a fast method for excluding malformed client requests (Section 5.5),
- a method to recover from transmission collisions in DC-net-style anonymity systems,
- experimental evaluation of these protocols with anonymity set sizes of up to 6,162,647 users (Section 5.6).

In Section 5.2, we introduce our goals, threat model, and security definitions. Section 5.3 presents the high-level system architecture. Section 5.4 and Section 5.5 detail our techniques for achieving bandwidth efficiency and disruption resistance in Riposte. We evaluate the performance of the system in Section 5.6, survey related work in Section 5.7, and conclude in Section 5.8.

5.2 Goals and problem statement

In this section, we summarize the high-level goals of the Riposte system and present our threat model and security definitions.

5.2.1 System goals

Riposte implements an anonymous bulletin board using a primitive we call a *write-private database scheme*. Riposte enables clients to write into a shared database, collectively maintained at a small set of servers, without revealing to the servers the location or contents of the write. Conceptually, the database table is just a long fixed-length bitstring divided into fixed-length rows.

To write into the database, a client generates a *write request*. The write request encodes the message to be written and the row index at which the client wants to write. (A single client write request modifies a single database row at a time.) Using cryptographic techniques, the client splits its write request into a number of shares and the client sends one share to each of the servers. By construction of the shares, no coalition of servers smaller than a particular pre-specified threshold can learn the contents of a single client's write request. While the cluster of servers must remain online for the duration of a protocol run, a client need only stay online for long enough to upload its write request to the servers. As soon as the servers receive a write request, they can apply it to to their local state.

The Riposte cluster divides time into a series of epochs. During each time epoch, servers collect many write requests from clients. When the servers agree that the epoch has ended, they combine their shares of the database to reveal the clients' plaintext messages. A particular client's anonymity set consists of all of the honest clients who submitted write requests to the servers during the time epoch. Thus, if 50,000 distinct honest clients submitted write requests during a particular time epoch, each honest client is perfectly anonymous amongst this set of 50,000 clients.

The epoch could be measured in time (e.g., 4 hours), in a number of write requests (e.g., accumulate 10,000 write requests before ending the epoch), or by some more complicated condition (e.g., wait for a write request signed from each of these 150 users identified by a pre-defined list of public keys). The definition of what constitutes an epoch is crucial for security, since a client's anonymity set is only as large as the number of honest clients who submit write requests in the same epoch [243].

When using Riposte as a platform for anonymous microblogging, the rows would be long enough to fit a Tweet (140 bytes) and the number of rows would be some multiple of the number of anticipated users. To anonymously Tweet, a client would use the write-private database scheme to write its message into a random row of the database. After many clients have written to the database, the servers can reveal the clients' plaintext Tweets. The write-privacy of the database scheme prevents eavesdroppers, malicious clients, and coalitions of malicious servers (smaller than a particular threshold) from learning which client posted which message.

5.2.2 Security properties

The Riposte system implements a *write-private* and *disruption-resistant* database scheme. We describe the correctness and security properties for such a scheme here.

Definition 5.2.1 (Correctness)**.** The scheme is *correct* if, when all servers execute the protocol faithfully, the plaintext state of the database revealed at the end of a protocol run is equal to the result of applying each valid client write requests to an empty database (i.e., a database of all zeros).

Notice that we only require correctness to hold when all servers execute the protocol faithfully. The failure—whether malicious or benign—of any one server renders the database state unrecoverable but *does not* compromise the anonymity of the clients. To protect against benign failures, server maintainers could implement a single "logical" Riposte server with a cluster of many physical servers running a standard state-machine-replication protocol [202, 222].

To be useful as an anonymous bulletin board, the database scheme must be *write-private* and *disruption resistant*. We define these security properties here.

Write privacy. Informally, the system provides *write-privacy* if an adversary—controlling as many as $s-1$ out of the s total servers and any number of clients—learns only the set of honest clients' messages, but not which client wrote which message. We define this property in terms of a simulator.

Definition 5.2.2 (Write privacy)**.** We say that an s-server scheme provides *write privacy* if for every efficient adversary, for every size-$(s-1)$ adversarial subset of the servers, and for

every choice of the honest clients' messages $(m_1, m_2, \dots)$, there exists a simulator S that takes as input

(i) the indices of the adversarial servers and clients and

(ii) the messages $(m_1, m_2, \dots)$ in lexicographic order

such that the following distributions are computationally indistinguishable:

- **Distribution $\mathcal{D}_{\mathbf{real}}$.** The view of the $s-1$ adversarial servers and clients, in an interaction with an honest server and honest clients submitting messages $(m_1, m_2, \dots)$ and
- **Distribution $\mathcal{D}_{\mathbf{ideal}}$.** The output of the simulator S, given the indices of the adversarial players and the messages $(m_1, m_2, \dots)$ in sorted order as input.

Riposte provides a very robust sort of privacy: the adversary can select the messages that the honest clients will send and can send maliciously formed messages that depend on the honest clients' messages. Even then, the adversary still cannot guess which client uploaded which message.

Furthermore, we make no assumptions about the behavior of malicious servers—they can misbehave by publishing their secret keys, by colluding with coalitions of up to $s-1$ malicious servers and arbitrarily many clients, or by mounting any other sort of attack against the system.

Disruption resistance. The system is *disruption resistant* if an adversary who controls n clients can write into at most n database rows during a single time epoch. A system that lacks disruption resistance might be susceptible to denial-of-service attacks: a malicious client could corrupt every row in the database with a single write request. Even worse, the write privacy of the system might prevent the servers from learning which client was the disruptor. Preventing such attacks is a major focus of prior anonymous messaging schemes [77, 141, 153, 267, 274]. We do not attempt to protect the operation of the system (i.e., correctness) in the face of misbehaving servers. Thus, our definition of disruption resistance concerns itself only with clients attempting to disrupt the system—we *do not* try to prevent servers from corrupting the database state. (In contrast, we *do* protect privacy in the face of misbehaving servers.)

We formally define *disruption resistance* using the following game, played between a challenger and an adversary. In this game, the challenger plays the role of all of the servers and the adversary plays the role of all clients.

1. The adversary sends n write requests to the challenger (where n is less than or equal to the number of rows in the database).
2. The challenger runs the protocol for a single time epoch, playing the role of the servers. The challenger then combines the servers' database shares to reveal the plaintext output.

The adversary wins the game if the plaintext output contains more than n non-zero rows.

Definition 5.2.3 (Disruption Resistance)**.** We say that the protocol is *disruption resistant* if the probability that the adversary wins the game above is negligible in the (implicit) security parameter.

5.2.3 Intersection Attacks

Riposte makes it infeasible for an adversary to determine which client posted which message *within* a particular time epoch. If an adversary can observe traffic patterns *across* many epochs, as the set of online clients changes, the adversary can make statistical inferences about which client is sending which stream of messages [108, 182, 206]. These "intersection" or "statistical disclosure" attacks affect many anonymity systems and defending against them is an important, albeit orthogonal, problem [206, 275]. Even so, intersection attacks typically become more difficult to mount as the size of the anonymity set increases, so Riposte's support for very large anonymity sets makes it less vulnerable to these attacks than are many prior systems.

5.3 System architecture

As described in the prior section, a Riposte deployment consists of a small number of servers, who maintain the database state, and a large number of clients. To write into the database, a client splits its write request using secret sharing techniques and sends a single share to each of the servers. Each server updates its database state using the client's share. After collecting write requests from many clients, the servers combine their shares to reveal the plaintexts represented by the write requests. The security requirement is that no coalition of t servers can learn which client wrote into which row of the database.

5.3.1 A first-attempt construction: Toy protocol

As a starting point, we sketch a simple "straw man" construction that demonstrates the techniques behind our scheme. This first-attempt protocol shares some design features with anonymous communication schemes based on client/server DC-nets [77, 274].

In the simple scheme, we have two servers, A and B, and each server stores an L-bit bitstring, initialized to all zeros. We assume for now that the servers *do not collude*—i.e., that one of the two servers is honest. The bitstrings represent shares of the database state and each "row" of the database is a single bit.

Consider a client who wants to write a "1" into row ℓ of the database. To do so, the client generates a random L-bit bitstring r. The client sends r to server A and $r \oplus e_\ell$ to server B, where e_ℓ is an L-bit vector of zeros with a one at index ℓ and $\oplus$ denotes bitwise XOR. Upon receiving the write request from the client, each server XORs the received string into its share of the database.

After processing n write requests, the database state at server A will be:

$$D_A = r_1 \oplus \cdots \oplus r_n \,,$$

and the database at server B will be:

$$\begin{aligned} D_B &= (e_{\ell_1} \oplus \cdots \oplus e_{\ell_n}) \oplus (r_1 \oplus \cdots \oplus r_n) \\ &= (e_{\ell_1} \oplus \cdots \oplus e_{\ell_n}) \oplus D_A \,. \end{aligned}$$

At the end of the time epoch, the servers can reveal the plaintext database by combining their local states D_A and D_B.

The construction generalizes to fields larger than $\mathbb{F}_2$. For example, each "row" of the database could be a k-bit bitstring instead of a single bit. To prevent impersonation, network-tampering, and replay attacks, we use authenticated and encrypted channels with per-message nonces bound to the time epoch identifier.

This protocol satisfies the write-privacy property as long as the two servers do not collude (assuming that the clients and servers deploy the replay attack defenses mentioned above). Indeed, server A can information theoretically simulate its view of a run of the protocol given only $e_{\ell_1} \oplus \cdots \oplus e_{\ell_n}$ as input. A similar argument shows that the protocol is write-private with respect to server B as well.

This first-attempt protocol has two major limitations. The first limitation is that it is not bandwidth-efficient. If millions of clients want to use the system in each time epoch, then the database must be at least millions of bits in length. To flip a *single bit* in the database then, each client must send *millions of bits* to each database, in the form of a write request.

The second limitation is that it is not disruption resistant: a malicious client can corrupt the entire database with a single malformed request. To do so, the malicious client picks random L-bit bitstrings r and r', sends r to server A, and sends r' (instead of $r \oplus e_\ell$) to server B. Thus, a single malicious client can efficiently and anonymously deny service to all honest clients.

Improving bandwidth efficiency and adding disruption resistance are the two core contributions of this work, and we return to them in Sections 5.4 and 5.5.

5.3.2 Collisions

Putting aside the issues of bandwidth efficiency and disruption resistance for the moment, we now discuss the issue of *colliding writes* to the shared database. If clients write into random locations in the database, there is some chance that one client's write request will overwrite a previous client's message. If client A writes message m_A into location ℓ, client B might later write message m_B into the same location ℓ. In this case, row ℓ will contain $m_A \oplus m_B$, and the contents of row ℓ will be unrecoverable.

To address this issue, we set the size of the database table to be large enough to accommodate the expected number of write requests for a given "success rate." For example, the servers can choose a table size that is large enough to accommodate 2^{10} write requests such that 95% of write requests will not be involved in a collision (in expectation). Under these parameters, 5% of the write requests will fail and those clients will have to resubmit their write requests in a future time epoch.

We can determine the appropriate table size by solving a simple "balls and bins" problem. If we throw n balls independently and uniformly at random into L bins, how many bins contain exactly one ball? Here, the n balls represent the write requests and the L bins represent the rows of the database.

Let B_{ij} be the probability that ball i falls into bin j. For all i and j, $\Pr[B_{ij}] = 1/L$. Let

$O_i^{(1)}$ be the event that *exactly* one ball falls into bin i. Then

$$\Pr\left[O_i^{(1)}\right] = \frac{n}{L}\left(1 - \frac{1}{L}\right)^{n-1}.$$

Expanding using the binomial theorem and ignoring low order terms we obtain

$$\Pr\left[O_i^{(1)}\right] \approx \frac{n}{L} - \left(\frac{n}{L}\right)^2 + \frac{1}{2}\left(\frac{n}{L}\right)^3,$$

where the approximation ignores terms of order $(n/L)^4$ and $o(1/L)$. Then $L \cdot \Pr[O_i^{(1)}]$ is the expected number of bins with exactly one ball which is the expected number of messages successfully received. Dividing this quantity by n gives the expected success rate so that:

$$\mathrm{E}[\text{SuccessRate}] = \frac{L}{n}\Pr[O_i^{(1)}] \approx 1 - \frac{n}{L} + \frac{1}{2}\left(\frac{n}{L}\right)^2.$$

So, if we want an expected success rate of 95% then we need $L \approx 19.5n$. For example, with $n = 2^{10}$ writers, we would use a table of size $L \approx 20{,}000$.

Handling collisions. We can shrink the table size L by coding the writes so that we can recover from collisions. We show how to handle two-way collisions, but the approach generalizes to higher-order collisions. That is, when at most two clients write to the same location in the database. Let us assume that the messages being written to the database are elements in some field $\mathbb{F}$ of odd characteristic (say $\mathbb{F} = \mathbb{F}_p$ where $p = 2^{64} - 59$). We replace the XOR operation used in the basic scheme by addition in $\mathbb{F}$.

To recover from a two-way collision we will need to double the size of each cell in the database, but the overall number of cells L will shrink by more than a factor of two.

When a client A wants to write the message $m_A \in \mathbb{F}$ to location ℓ in the database the client will actually write the pair $(m_A, m_A^2) \in \mathbb{F}^2$ into that location. Clearly if no collision occurs at location ℓ then recovering m_A at the end of the epoch is trivial: simply drop the second coordinate (it is easy to test that no collision occurred because the second coordinate is a square of the first). Now, suppose a collision occurs with some client B who also added her own message $(m_B, m_B^2) \in \mathbb{F}^2$ to the same location ℓ (and no other client writes to location ℓ). Then at the end of the epoch the published values are

$$S_1 = m_A + m_B \in \mathbb{F} \qquad \text{and} \qquad S_2 = m_A^2 + m_B^2 \in \mathbb{F}.$$

From these values it is then possible to recover both m_A and m_B by observing that

$$2S_2 - S_1^2 = (m_A - m_B)^2 \quad \in \mathbb{F}$$

from which we obtain $m_A - m_B$ by taking a square root in $\mathbb{F}$. (It does not matter which of the two square roots we use—they both lead to the same result.) Given $S_1 = m_A + m_B \in \mathbb{F}$, it is possible to recover both m_A and m_B.

Now that we can recover from two-way collisions we can shrink the number of cells L in the table. Let $O_i^{(2)}$ be the event that exactly two balls fall into bin i. Then the expected number of received messages is

$$L \Pr[O_i^{(1)}] + 2L \Pr[O_i^{(2)}], \tag{5.1}$$

where $\Pr[O_i^{(2)}] = \binom{n}{2} \frac{1}{L^2} \left(1 - \frac{1}{L}\right)^{n-2}$. As before, dividing the expected number of received messages (5.1) by n, expanding using the binomial theorem, and ignoring low order terms gives the expected success rate as:

$$\mathrm{E}[\text{SuccessRate}] \approx 1 - \frac{1}{2}\left(\frac{n}{L}\right)^2 + \frac{1}{3}\left(\frac{n}{L}\right)^3 .$$

So, if we want an expected success rate of 95% we need a table with $L \approx 2.7n$ cells. This is a far smaller table than before, when we could not handle collisions. In that case we needed $L \approx 19.5n$ which results in much bigger tables, despite each cell being half as big. Shrinking the table reduces the storage and computational burden on the servers.

This two-way collision handling technique generalizes to handle κ-way collisions for $\kappa > 2$. To handle κ-way collisions, we increase the size of each cell by a factor of κ and have each client i write $(m_i, m_i^2, \ldots, m_i^\kappa) \in \mathbb{F}^\kappa$ to its chosen cell. A κ-collision gives κ equations in κ variables that we can efficiently solve to recover all κ messages, as long as the characteristic of $\mathbb{F}$ is greater than κ [61, 82]. Using $\kappa > 2$ further reduces the table size as the desired success rate approaches one.

The collision handling method described in this section will also improve performance of our full system, which we describe in the next section.

Adversarial collisions. The analysis above assumes that clients behave honestly. Adversarial clients, however, need not write into random rows of the database—i.e., all n balls might not

be thrown independently and uniformly at random. A coalition of clients might, for example, try to increase the probability of collisions by writing into the database using some malicious strategy.

By symmetry of writes we can assume that all $\hat{n}$ adversarial clients write to the database before the honest clients do. Now a message from an honest client is properly received at the end of an epoch if it avoids all the cells filled by the malicious clients. We can therefore carry out the honest client analysis above assuming the database contain $L - \hat{n}$ cells instead of L cells. In other words, given a bound $\hat{n}$ on the number of malicious clients we can calculate the required table size L. In practice, if the servers detect too many collisions at the end of an epoch, they can adaptively double the size of the table so that the next epoch has fewer collisions.

5.3.3 Forward security

Even the first-attempt scheme sketched in Section 5.3.1 provides *forward security* in the event that all of the servers' secret keys are compromised [71]. To be precise: an adversary could compromise the state and secret keys of *all servers* after the servers have processed n write requests from honest clients, but *before* the time epoch has ended. Even in this case, the adversary will be unable to determine which of the n clients submitted which of the n plaintext messages with a non-negligible advantage over random guessing. (We assume here that clients and servers communicate using encrypted channels which themselves have forward secrecy [181].)

This forward security property means that clients need not trust the one honest server to stay honest forever—just that it is honest at the moment at which the client submits its upload request. Being able to weaken the trust assumption about the servers in this way might be valuable in hostile environments, in which an adversary could compromise a server at any time without warning.

Mix-nets do not have this property, since servers must accumulate a set of onion-encrypted messages before shuffling and decrypting them [78]. If an adversary always controls the first mix server and if it can compromise the rest of the mix servers after accumulating a set of ciphertexts, the adversary can de-anonymize all of the system's users. DC-net-based systems that use "blame" protocols to retroactively discover disruptors have a similar weakness [95, 274].

The full Riposte protocol maintains this forward security property.

5.4 Reducing communication with distributed point functions

This section describes how application of private information retrieval techniques can improve the bandwidth efficiency of the first-attempt protocol.

5.4.1 Definitions

The bandwidth inefficiency of the toy protocol sketched in Section 5.3.1 comes from the fact that the client must send an L-bit vector to each server to flip a single bit in the logical database. To reduce this $O(L)$ bandwidth overhead, we apply techniques inspired by private information retrieval protocols [84, 86, 138].

The problem of private information retrieval (PIR) is essentially the converse of the problem we are interested in here. In PIR, the client must *read* a bit from a replicated database without revealing to the servers the index being read. In our setting, the client must *write* a bit into a replicated database without revealing to the servers the index being written. Ostrovsky and Shoup first made this connection in the context of a "private information storage" protocol [223].

PIR schemes allow the client to split its query to the servers into shares such that (1) a proper subset of the shares does not leak information about the index of interest, and (2) the length of the query shares is much less than the length of the database. The core building block of many PIR schemes, which we adopt for our purposes, is a *distributed point function*. Although Gilboa and Ishai [138] defined distributed point functions as a primitive only recently, many prior PIR schemes make implicit use the primitive [84, 86]. Our definition of a distributed point function follows that of Gilboa and Ishai, except that we generalize the definition to allow for more than two servers.

First, we define a (non-distributed) point function.

Definition 5.4.1 (Point function)**.** Fix a positive integer L and a finite field $\mathbb{F}$. For all $\ell \in [L]$ and $m \in \mathbb{F}$, the *point function* $P_{\ell,m} : [L] \to \mathbb{F}$ is the function such that $P_{\ell,m}(\ell) = m$ and $P_{\ell,m}(\ell') = 0$ for all $\ell \neq \ell'$.

That is, the point function $P_{\ell,m}$ has the value 0 when evaluated at any input not equal to ℓ and it has the value m when evaluated at ℓ. For example, if $L = 5$ and $\mathbb{F} = \mathbb{F}_2$, the point function $P_{4,1}$ takes on the values $(0, 0, 0, 1, 0)$ when evaluated on the values $(1, 2, 3, 4, 5)$, where we index vectors from one.

An (s, t)-distributed point function provides a way to distribute a point function $P_{\ell,m}$ amongst s servers such that no coalition of at most t servers learns anything about ℓ or m given their t shares of the function.

Definition 5.4.2 (Distributed point function, DPF [138])**.** Fix a positive integer L and a finite field $\mathbb{F}$. An (s, t)-*distributed point function* consists of a pair of possibly randomized algorithms that implement the following functionalities:

- $\mathsf{Gen}(\ell, m) \to (k_1, \ldots, k_s)$. Given an integer $\ell \in [L]$ and value $m \in \mathbb{F}$, output a list of s keys.
- $\mathsf{Eval}(k, \ell') \to m'$. Given a key k generated using Gen, and an index $\ell' \in [L]$, return a value $m' \in \mathbb{F}$.

We define correctness and privacy for a distributed point function as follows:

- **Correctness.** For a collection of s keys generated using $\mathsf{Gen}(\ell, m)$, the sum of the outputs of these keys (generated using Eval) must equal the point function $P_{\ell,m}$. More formally, for all $\ell, \ell' \in [L]$ and $m \in \mathbb{F}$:

$$\Pr\left[\left(\sum_{i\in[s]} \mathsf{Eval}(k_i, \ell')\right) = P_{\ell,m}(\ell') \colon (k_1, \ldots, k_s) \leftarrow \mathsf{Gen}(\ell, m)\right] = 1,$$

  where the probability is taken over the randomness of the Gen algorithm.

- **Privacy.** Let S be any subset of $[s]$ such that $|S| \leq t$. Then for any $\ell \in [L]$ and $m \in \mathbb{F}$, let $D_{S,\ell,m}$ denote the distribution of keys $\{(k_i) \mid i \in S\}$ induced by $(k_1, \ldots, k_s) \leftarrow \mathsf{Gen}(\ell, m)$. We say that an (s, t)-DPF maintains privacy if there exists an efficient algorithm Sim such that for all choice of S, ℓ, and m, the following distributions are computationally indistinguishable:

$$D_{S,\ell,m} \approx_c \mathsf{Sim}(S)$$

  Informally, any subset of at most t keys leaks no information about ℓ or m.

Toy construction. To make this definition concrete, we first construct a trivial information-theoretically secure $(s, s-1)$-distributed point function with keys of length $O(L \cdot \log|\mathbb{F}|)$ bits. As above, we fix a length L and a finite field $\mathbb{F}$.

- $\mathsf{Gen}(\ell, m) \to (k_1, \ldots, k_s)$. Sample s random vectors $k_1, \ldots, k_s \in \mathbb{F}^L$ subject to the

constraint that

$$m \cdot e_\ell = \sum_{i \in [s]} k_i \quad \in \mathbb{F}^L.$$

- $\mathsf{Eval}(k, \ell') \to m'$. Interpret k as a vector in $\mathbb{F}^L$. Return the value of the vector k at index ℓ'.

The correctness property of this construction follows immediately. Privacy is maintained because the distribution of any collection of $s - 1$ keys is independent of ℓ and m.

This toy construction uses length-$\Omega(L)$ keys to distribute a point function with domain $[L]$. Later in this section we describe DPF constructions which use much shorter keys.

5.4.2 Applying distributed point functions for bandwidth efficiency

We can now use DPFs to improve the efficiency of the write-private database scheme introduced in Section 5.3.1. We show that the existence of an (s, t)-DPF with keys of length $|k|$ (along with standard cryptographic assumptions) implies the existence of write-private database scheme using s servers that maintains anonymity in the presence of t malicious servers, such that write requests have length $s|k|$. Any DPF construction with short keys thus immediately implies a bandwidth-efficient write-private database scheme.

The construction is a generalization of the one presented in Section 5.3.1. We now assume that there are s servers such that no more than t of them collude. Each of the s servers maintains a vector in $\mathbb{F}^L$ as their database state, for some fixed finite field $\mathbb{F}$ and integer L. Each "row" in the database is now an element of $\mathbb{F}$ and the database has L rows.

When the client wants to write a message $m \in \mathbb{F}$ into location $\ell \in [L]$ in the database, the client uses an (s, t)-distributed point function to generate a set of s DPF keys:

$$(k_1, \ldots, k_s) \leftarrow \mathsf{Gen}(\ell, m)$$

The client then sends one of the keys to each of the servers. Each server i can then expand the key into a vector $v_i \in \mathbb{F}^L$ by computing

$$v_i \leftarrow \big(\mathsf{Eval}(k_i, 1), \mathsf{Eval}(k_i, 2), \ldots, \mathsf{Eval}(k_i, L)\big) \quad \in \mathbb{F}^L.$$

The server then adds this vector v_i into its database state, using addition in $\mathbb{F}^L$. At the end of the time epoch, all servers combine their database states to reveal the set of client-submitted

messages.

Correctness. The correctness of this construction follows directly from the correctness of the DPF. For each of the n write requests submitted by the clients, denote the j-th key in the i-th request as $k_{i,j}$, denote the write location as ℓ_i, and the message being written as m_i. When the servers combine their databases at the end of the epoch, the contents of the final database at row ℓ will be:

$$D_\ell = \sum_{i \in [n]} \sum_{j \in [s]} \mathsf{Eval}(k_{i,j}, \ell) = \sum_{i \in [n]} P_{\ell_i, m_i}(\ell) \quad \in \mathbb{F}$$

In words: as desired, the combined database contains the sum of n point functions—one for each of the write requests.

Write privacy. The write privacy of this construction follows directly from the privacy property of the DPF. Given the plaintext database state D (as defined above), any coalition of t servers can simulate its view of the protocol. By definition of DPF privacy, there exists a simulator Sim, which simulates the distribution of any subset of t DPF keys generated using Gen. The coalition of servers can use this simulator to simulate each of the n write requests it sees during a run of the protocol, given only the final set of messages as input.

Efficiency. A client in this scheme sends $|k|$ bits to each server (where k is a DPF key), so the bandwidth efficiency of the scheme depends on the efficiency of the DPF. As we will show later in this section, $|k|$ can be much smaller than the length of the database.

5.4.3 A two-server scheme tolerating one malicious server

Having established that DPFs with short keys lead to bandwidth-efficient write-private database schemes, we now present one such DPF construction. This construction is a simplification of the computational PIR scheme of Chor and Gilboa [84].

This is a $(2, 1)$-DPF with keys of length $O(\sqrt{L})$ operating on a domain of size L. (Here, and throughout this section, the key lengths hide fixed polynomials in the security parameter and the field size.) This DPF yields a two-server write-private database scheme tolerating one malicious server such that writing into a database of size L requires sending $O(\sqrt{L})$ bits to each server. Gilboa and Ishai [138] construct a $(2, 1)$-DPF with even shorter keys of length $\text{polylog}(L)$. Later work work [62, 63] reduces the key size even further, to the optimal

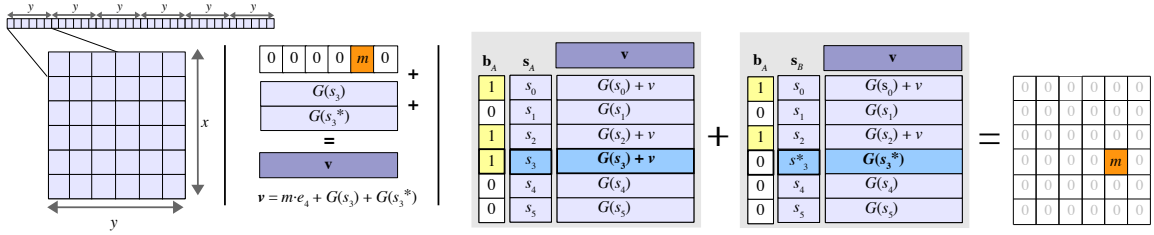


Figure 5.1: Left: We represent the output of Eval as an $x \times y$ matrix of field elements. Left-center: Construction of the $\mathbf{v}$ vector used in the DPF keys. Right: using the $\mathbf{v}$, $\mathbf{s}$, and $\mathbf{b}$ vectors, Eval expands each of the two keys into an $x \times y$ matrix of field elements. These two matrices sum to zero everywhere except at $(\ell_x, \ell_y) = (4, 5)$, where they sum to m.

$|k| = O(\log L)$, but the construction presented here is efficient enough for the database sizes we use in practice. For ease of exposition, we describe the DPF construction using a finite field $\mathbb{F} = \mathbb{F}_{2^\beta}$ of characteristic two, though the construction actually only requires working over a cyclic group.

When we run $\mathsf{Eval}(k, \ell')$ on every integer $\ell' \in [L]$, its output is a vector of L field elements. The DPF key construction conceptually works by representing this a vector of L field elements as an $x \times y$ matrix, such that $xy \geq L$. The trick that makes the construction work is that the size of the keys needs only to grow with the size of the *sides* of this matrix rather than its *area*. The DPF keys that the $\mathsf{Gen}(\ell, m)$ routine outputs give an efficient way to construct two matrices M_1 and M_2 that differ only at one cell $\ell = (\ell_x, \ell_y) \in [x] \times [y]$ (Figure 5.1).

Fix a binary finite field $\mathbb{F} = \mathbb{F}_{2^\beta}$, a DPF domain size L, and integers x and y such that $xy \geq L$. (Later in this section, we describe how to choose x and y to minimize the key size.) The construction requires a pseudo-random generator (PRG) that stretches seeds from $\mathbb{S}$ into length-y vectors of elements of $\mathbb{F}$. So the type signature of the PRG is $\mathsf{PRG} : \mathbb{S} \to \mathbb{F}^y$. In practice, an implementation might use AES-128 in counter mode as the pseudo-random generator [219].

The algorithms comprising the DPF, on domain size L, field $\mathbb{F}$, and key space $\mathcal{K}$, are:

- $\mathsf{Gen}(\ell \in [\ell], m \in \mathbb{F}) \to (k_1, k_2) \in \mathcal{K}^2$.
  - Compute integers $\ell_x \in [x]$ and $\ell_y \in [y]$ such that $\ell = \ell_x y + \ell_y$.
  - Sample a random bit-vector $\mathbf{b}_1 \xleftarrow{R} \{0,1\}^x$, a random vector of PRG seeds $\boldsymbol{\sigma}_1 \xleftarrow{R} \mathbb{S}^x$.
  - Sample a single random PRG seed $\sigma^*_{\ell_x} \xleftarrow{R} \mathbb{S}$.

– Given $\mathbf{b}_1$ and $\boldsymbol{\sigma}_1$, we define $\mathbf{b}_2$ and $\boldsymbol{\sigma}_2$ as:

$$\begin{aligned}
\mathbf{b}_1 &= (b_1, \ldots, b_{\ell_x}, \ldots, b_x) && \in \{0,1\}^x \subseteq \mathbb{F}^x \\
\mathbf{b}_2 &= (b_1, \ldots, 1 - b_{\ell_x}, \ldots, b_x) && \in \{0,1\}^x \subseteq \mathbb{F}^x \\
\boldsymbol{\sigma}_1 &= (\sigma_1, \ldots, \sigma_{\ell_x}, \ldots, \sigma_x) && \in \mathbb{S}^x \\
\boldsymbol{\sigma}_2 &= (\sigma_1, \ldots, \sigma^*_{\ell_x}, \ldots, \sigma_x) && \in \mathbb{S}^x
\end{aligned}$$

That is, the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ (similarly $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$) differ only at index ℓ_x.

– Let $m \cdot e_{\ell_y}$ be the vector in $\mathbb{F}^y$ of all zeros except that it has value $m \in \mathbb{F}$ at index ℓ_y.

– Define $\mathbf{v} \leftarrow m \cdot e_{\ell_y} + \mathsf{PRG}(\sigma_{\ell_x}) + \mathsf{PRG}(\sigma^*_{\ell_x}) \in \mathbb{F}^y$.

The output DPF keys are:

$$k_1 = (\mathbf{b}_1, \boldsymbol{\sigma}_1, \mathbf{v}) \qquad \text{and} \qquad k_2 = (\mathbf{b}_2, \boldsymbol{\sigma}_2, \mathbf{v}).$$

- $\mathsf{Eval}(k \in \mathcal{K}, \ell' \in [L]) \to m' \in \mathbb{F}$. Interpret the key k as a tuple $(\mathbf{b}, \boldsymbol{\sigma}, \mathbf{v})$. To evaluate the PRF at index ℓ', first write ℓ' as an (ℓ'_x, ℓ'_y) tuple such that $\ell'_x \in [x]$, $\ell'_y \in [y]$, and $\ell' = \ell'_x y + \ell'_y$.

  Let $\mathsf{PRGMatrix} : \mathbb{S}^x \to \mathbb{F}^{x \times y}$ be the operator that applies the PRG $\mathsf{PRG} : \mathbb{S} \to \mathbb{F}^y$ to each of the x seeds given as input to form a matrix in $\mathbb{F}^{x \times y}$. Construct the matrix

$$M = \mathbf{b} \cdot \mathbf{v}^T + \mathsf{PRGMatrix}(\boldsymbol{\sigma}) \quad \in \mathbb{F}^{x \times y}$$

  and output the element at index (ℓ'_x, ℓ'_y). Notice that it is possible to construct this single element without having to generate the entire matrix.

Figure 5.1 graphically depicts how Eval stretches the keys into a table of $x \times y$ field elements.

Correctness. For the scheme to be correct, it must be that, for all $\ell \in [L]$ and $m \in \mathbb{F}$, when we generate a pair of keys as $(k_1, k_2) \leftarrow \mathsf{Gen}(\ell, m)$, it holds that for all $\ell' \in [L]$

$$\mathsf{Eval}(k_1, \ell') + \mathsf{Eval}(k_2, \ell') = P_{\ell,m}(\ell') \quad \in \mathbb{F}.$$

Then, consider the matrix $M_1 \in \mathbb{F}^{x \times y}$ whose entry at coordinate (ℓ_x, ℓ_y) is the value

$\mathsf{Eval}(k_a, \ell_x \cdot y + \ell_y) \in \mathbb{F}$. By construction of the DPF, we can write

$$\begin{aligned} M_1 &= \mathbf{b}_1 \cdot \mathbf{v}^T + \mathsf{PRGMatrix}(\boldsymbol{\sigma}_1) \quad \in \mathbb{F}^{x \times y} \\ M_2 &= \mathbf{b}_2 \cdot \mathbf{v}^T + \mathsf{PRGMatrix}(\boldsymbol{\sigma}_2) \quad \in \mathbb{F}^{x \times y}. \end{aligned}$$

Then, using the fact that $\mathbb{F}$ has characteristic two so that, for all $\alpha \in \mathbb{F}$, it holds that $\alpha + \alpha = 0 \in \mathbb{F}$:

$$\begin{aligned} M_1 + M_2 &= (\mathbf{b}_1 + \mathbf{b}_2) \cdot \mathbf{v}^T + \mathsf{PRGMatrix}(\boldsymbol{\sigma}_1) + \mathsf{PRGMatrix}(\boldsymbol{\sigma}_2) \quad \in \mathbb{F}^{x \times y} \\ &= e_{\ell_x} \cdot \mathbf{v}^T + e_{\ell_x} \cdot \left(\mathsf{PRG}(\sigma_{\ell_x}) + \mathsf{PRG}(\sigma^*_{\ell_x})\right)^T \\ &= e_{\ell_x} \cdot \left(\mathbf{v} + \mathsf{PRG}(\sigma_{\ell_x}) + \mathsf{PRG}(\sigma^*_{\ell_x})\right)^T \\ &= m \cdot e_{\ell_x} \cdot e_{\ell_y}^T. \end{aligned}$$

Therefore, the sum of the evaluations of the two DPF keys at all points in $[x] \times [y]$ is $0 \in \mathbb{F}$, except that at the special point (ℓ_x, ℓ_y), this sum takes on value $m \in \mathbb{F}$, as required.

Privacy. The privacy property requires that there exists an efficient simulator that, on input "1" or "2," outputs samples from a distribution that is computationally indistinguishable from the distribution of DPF keys k_1 or k_2, generated using some index $\ell \in [L]$ and message $m \in \mathbb{F}$.

The simulator Sim simulates each component of the DPF key as follows: It samples $\mathbf{b} \xleftarrow{R} \{0,1\}^x$, $\boldsymbol{\sigma} \xleftarrow{R} \mathbb{S}^x$, and $\mathbf{v} \xleftarrow{R} \mathbb{F}^y$. The simulator returns $(\mathbf{b}, \boldsymbol{\sigma}, \mathbf{v})$.

We must now argue that the simulator's output distribution is computationally indistinguishable from that induced by the distribution of a single output of Gen. To do so, we show that we can use any adversary that distinguishes these distributions to break the underlying PRG. In one hybrid step, we replace the vector $\mathbf{v}$, output by Gen, with a random vector. Any adversary that can distinguish these hybrid distributions can break the underlying PRG.

Key size. A key for this DPF scheme consists of: a vector in $\{0,1\}^x$, a vector in $\mathbb{S}^x$, and a vector in $\mathbb{F}^y$. Let α be the number of bits required to represent an element of $\mathbb{S}$ and let β be the number of bits required to represent an element of $\mathbb{F}$. The total length of a key is then:

$$|k| = (1 + \alpha)x + \beta y$$

For fixed spaces $\mathbb{S}$ and $\mathbb{F}$, we can find the optimal choices of x and y to minimize the key

length. To do so, we solve:

$$\min_{x,y}((1+\alpha)x + \beta y) \quad \text{subject to} \quad xy \geq L$$

and conclude that the optimal values of x and y are:

$$x = c\sqrt{L} \quad \text{and} \quad y = \frac{1}{c}\sqrt{L} \quad \text{where} \quad c = \sqrt{\frac{\beta}{1+\alpha}}.$$

The key size is then $O(\sqrt{L})$.

When using a database table of one million rows in length ($L = 2^{20}$), a row length of 1 KB per row ($\mathbb{F} = \mathbb{F}_{2^{8192}}$), and a PRG seed size of 128 bits (using AES-128, for example) the keys will be roughly 263 KB in length. For these parameters, the keys for the naïve construction (Section 5.3.1) would be 1 GB in length. Application of efficient DPFs thus yields a 4,000× bandwidth savings in this case.

Computational efficiency. A second benefit of this scheme is that both the Gen and Eval routines are concretely efficient, since they just require performing finite field additions (i.e., XOR for binary fields) and PRG operations (i.e., computations of the AES function). The construction requires no public-key primitives.

5.4.4 An s-server scheme tolerating $s-1$ malicious servers

The $(2,1)$-DPF scheme described above achieved a key size of $O(\sqrt{L})$ bits using only pseudorandom generators—i.e., symmetric-key primitives. (Again, the big-O hides fixed polynomials in the security parameter and underlying field size.) The limitation of that construction is that it only maintains privacy when a single key is compromised. In the context of a write-private database scheme, this means that the construction can only maintain anonymity in the presence of a *single* malicious server. It would be much better to have a write-private database scheme with s servers that maintains anonymity in the presence of $s-1$ malicious servers. To achieve this stronger security notion, we need a bandwidth-efficient $(s, s-1)$-distributed point function.

In this section, we construct an $(s, s-1)$-DPF in which each key has size $O(\sqrt{L})$. We do so at the cost of requiring more expensive public-key cryptographic operations, instead of the symmetric-key operations used in the prior DPF. While the $(2,1)$-DPF construction above directly follows the work of Chor and Gilboa [84], this $(s, s-1)$-DPF construction is

new. In recent work, Boyle et al. present an $(s, s-1)$-DPF construction from psuedorandom generators, but this construction exhibits a key size *exponential* in the number of servers s [62]. In contrast, the key sizes in our construction are independent of the number of servers s.

This construction uses a *seed-homomorphic pseudorandom generator* [20, 57, 218], to split the key for the pseudo-random generator PRG across a collection of s DPF keys.

We will need a group $\mathbb{G}$ of prime order q. When $\mathbf{u} = (u_1, \ldots, u_y)$ and $\mathbf{v} = (v_1, \ldots, v_y)$ are vectors in $\mathbb{G}^y$, we denote the component-wise product in $\mathbb{G}$ of $\mathbf{u}$ and $\mathbf{v}$ as

$$\mathbf{u} \circ \mathbf{v} = (u_1 v_1, \ldots, u_y v_y) \quad \in \mathbb{G}^y.$$

Furthermore, for a vector of exponents $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_y) \in \mathbb{Z}_y$ and a vector of group elements $\mathbf{u} = (u_1, \ldots, u_y) \in \mathbb{G}^y$, we use the shorthand

$$\mathbf{u}^{\boldsymbol{\sigma}} = (u_1^{\sigma_1}, \ldots, u_y^{\sigma_y}) \quad \in \mathbb{G}^y.$$

When $m \in \mathbb{Z}_q$ is a scalar, we define

$$\mathbf{u}^m = (u_1^m, \ldots, u_y^m) \quad \in \mathbb{G}^y.$$

Definition 5.4.3 (Seed-homomorphic PRG)**.** A *seed-homomorphic* PRG is a pseudo-random generator $\mathsf{PRG}\colon \mathbb{Z}_q \to \mathbb{G}^y$ with the additional property that for any pair of seeds $s_1, s_2 \in \mathbb{Z}_q$:

$$\mathsf{PRG}(s_1 + s_1 \in \mathbb{Z}_q) = \mathsf{PRG}(s_1) \circ \mathsf{PRG}(s_2) \quad \in \mathbb{G}$$

We require also that $\mathsf{PRG}(0)$ is equal to the identity element in $\mathbb{G}$.

It is possible to construct a simple seed-homomorphic PRG from algebraic groups in which the Decision Diffie-Hellman (DDH) assumption holds [57, 218]. The public parameters for the scheme consist of a list of y randomly sampled generators $(g_1, \ldots, g_y) \in \mathbb{G}^y$ of a cyclic group $\mathbb{G}$, in which the DDH problem is hard [53]. Given a seed $\sigma \in \mathbb{Z}_q$, the generator outputs $(g_1^\sigma, \ldots, g_y^\sigma)$. The generator is seed-homomorphic because, for any $\sigma_1, \sigma_2 \in \mathbb{Z}_q$ and $g \in \mathbb{G}$, $g^{\sigma_1} g^{\sigma_2} = g^{(\sigma_1 + \sigma_2)} \in \mathbb{G}$.

Messages "in the exponent." If we instantiate the seed-homomorphic PRG with the DDH-based construction in a group $\mathbb{G}$ of prime order q, the message space of our DPF

is is $\mathbb{Z}_q$ and the DPF Eval routine outputs elements in $\mathbb{G}$. More precisely, for a fixed generator $g \in \mathbb{G}$, the DPF correctness property now states that for all $\ell \in [L]$, $m \in \mathbb{Z}_q$, and $(k_1, \ldots, k_s) \leftarrow \mathsf{Gen}(\ell, m)$,

$$\sum_{i \in [s]} \mathsf{Eval}(k_i, \ell') = \begin{cases} g^m & \text{if } \ell' = \ell \\ g^0 & \text{otherwise} \end{cases} \in \mathbb{G}.$$

The fact that the DPF's output is $g^m \in \mathbb{G}$ means that it is not possible to recover m directly from the DPF output if m can be arbitrary in $\mathbb{Z}_q$, since recovering m from g^m requires solving the discrete-logarithm problem in $\mathbb{G}$. If we require $|m|$ to bounded by a fixed polynomial in $\log |\mathbb{G}|$, then it is possible to recover m by brute force.

It is actually also possible to modify our construction to have message space $\mathbb{G}$, which avoids this problem. But then our disruption-resistance techniques of Section 5.5.2 do not apply. In any event, it is possible to construct a DPF with a large message space by running many instances of this DPF construction in parallel.

The DPF construction. As in the prior DPF construction, we fix a DPF domain size L, and integers x and y such that $xy \geq L$. The construction requires a group $\mathbb{G}$ of prime order q with a fixed generator g and a seed-homomorphic PRG $\mathsf{PRG} : \mathbb{Z}_q \mapsto \mathbb{G}^y$.

The algorithms comprising the $(s, s-1)$-DPF, with key space $\mathcal{K}$, are:

- $\mathsf{Gen}(\ell \in [L], m \in \mathbb{Z}_q) \to (k_1, \ldots, k_s) \in \mathcal{K}^s$.
  - Compute integers $\ell_x \in [x]$ and $\ell_y \in [y]$ such that $\ell = \ell_x y + \ell_y$.
  - Sample random vectors $\mathbf{b}_1, \ldots, \mathbf{b}_s \in \mathbb{Z}_q^x$ such that $\sum_{i \in [s]} \mathbf{b}_i = e_{\ell_x} \in \mathbb{Z}_q^x$.
  - Sample a single random PRG seed $\sigma^* \xleftarrow{R} \mathbb{G}$.
  - Sample random vectors of PRG seeds $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_s \in \mathbb{Z}_q^x$ such that $\sum_{i \in [s]} = \sigma^* \cdot e_{\ell_x} \in \mathbb{Z}_q^x$.
  - Define $\mathbf{v} \leftarrow e_{\ell_y}^m \circ \mathsf{PRG}(\sigma^*)^{-1} \in \mathbb{G}^y$.

    Here, $e_{\ell_y} \in \mathbb{G}^y$ is the vector in $\mathbb{G}^y$ that consists of the identity element $1_{\mathbb{G}}$ everywhere, except that it has the generator $g \in \mathbb{G}$ at the ℓ_y-th coordinate. More explicitly, the vector $e_{\ell_y}^m$ takes the form:

$$e_{\ell_y}^m = (g^0, g^0, \ldots, g^0, \underbrace{g^m}_{\ell_y\text{-th coordinate}}, g^0, \ldots, g^0) \quad \in \mathbb{G}^y.$$

The DPF key for server $i \in [s]$ is $k_i = (\mathbf{b}_i, \boldsymbol{\sigma}_i, \mathbf{v})$.

- $\mathsf{Eval}(k, \ell') \to m'$. Interpret k as a tuple $(\mathbf{b}, \boldsymbol{\sigma}, \mathbf{v})$.

  As in the prior DPF construction, define an operator $\mathsf{PRGMatrix} : \mathbb{Z}_q^x \to \mathbb{G}^{x \times y}$ that applies the PRG $\mathsf{PRG} : \mathbb{Z}_q \to \mathbb{G}^y$ to each of the x seeds given as input to form a matrix in $\mathbb{G}^{x \times y}$.

  To evaluate the PRF at index ℓ', first write ℓ' as an (ℓ'_x, ℓ'_y) tuple such that $\ell'_x \in [x]$, $\ell'_y \in [y]$, and $\ell' = \ell'_x y + \ell'_y$. Then let $\mathbf{b} = (b_1, \ldots, b_x) \in \mathbb{Z}_q$ and compute the matrix

$$M = \mathsf{PRGMatrix}(\boldsymbol{\sigma}) \circ \begin{pmatrix} \mathbf{v}^{b_1} \\ \mathbf{v}^{b_2} \\ \vdots \\ \mathbf{v}^{b_x} \end{pmatrix} \quad \in \mathbb{G}^{x \times y}$$

  and output the element of this matrix at index (ℓ'_x, ℓ'_y).

We omit correctness and privacy proofs, since they follow exactly the same structure as those used to prove security of our prior DPF construction. The only difference is that correctness here relies on the fact that PRG is a seed-homomorphic PRG, rather than a conventional PRG. As in the DPF construction of Section 5.4.3, the keys here are of length $O(\sqrt{L})$.

Computational efficiency. The main computational cost of this DPF construction comes from the use of the seed-homomorphic PRG. *Unlike* a conventional PRG, which can be implemented using AES or another fast block cipher in counter mode, known constructions of seed-homomorphic PRGs require algebraic groups [218] or lattice-based cryptography [20, 57].

When instantiating the $(s, s-1)$-DPF with the DDH-based PRG construction in elliptic curve groups, each call to the DPF Eval routine requires an expensive elliptic curve scalar multiplication. Since elliptic curve operations are, per byte, orders of magnitude slower than AES operations, this $(s, s-1)$-DPF will be orders of magnitude slower than the $(2, 1)$-DPF. Security against an arbitrary number of malicious servers comes at the cost of computational efficiency, at least for these DPF constructions.

With DPFs, we can now construct a bandwidth-efficient write-private database scheme that tolerates one malicious server, using the DPF of Section 5.4.3, or $s-1$ out of s malicious

servers, using the DPF of Section 5.4.4.

5.5 Preventing disruptors

The first-attempt construction of our write-private database scheme (Section 5.3.1) had two limitations: (1) client write requests were very large and (2) malicious clients could corrupt the database state by sending malformed write requests. We addressed the first of these two challenges in Section 5.4. In this section, we address the second challenge.

A client write request in our protocol just consists of a collection of s DPF keys. The client sends one key to each of the s servers. The servers must collectively decide whether the collection of s keys is a valid output of the DPF Gen routine, without revealing any information about the keys themselves.

One way to view the servers' task here is as a secure multi-party computation [146, 277]. Each server i's private input is its DPF key k_i. The output of the protocol is a single bit that indicates whether the s keys $(k_1, \ldots, k_s)$ are a well-formed collection of DPF keys.

Since we require correctness to hold only if all servers are honest (Section 5.2.2), we *need not* protect against servers maliciously trying to manipulate the output of the multi-party protocol. Such manipulation could only result in corrupting the database (if a malicious server accepts a write request that it should have rejected) or denying service to an honest client (if a malicious server rejects a write request that it should have accepted). Since both attacks are tantamount to denial of service, we need not consider them.

We *do* care, in contrast, about protecting client privacy against malicious servers. A malicious server participating in the protocol should not gain any additional information about the private inputs of other parties, no matter how it deviates from the protocol specification.

An early version of this work [94] proposed protecting against disruptors using special-purpose multi-party protocols or discrete-log-based zero-knowledge proofs. The technique based on multi-party computation was undesirable because it required *three* servers such that *no two* colluded. (In contrast, the DPF construction itself only requires two non-colluding servers total.) The technique based on general-purpose zero-knowledge proofs was undesirable because it required many expensive public-key cryptographic operations to check a single client submission.

By applying the new results of Chapter 3, we eliminate these limitations: we can protect against disruptors in Riposte while needing only two servers and while relying only on lightweight symmetric-key techniques (pseudorandom generators). We use these new techniques in recent work on metadata-hiding messaging [122].

5.5.1 The two-server setting: Proofs on secret-shared data

We cast the problem of DPF key checking in the new language of zero-knowledge proofs on secret-shared data (Chapter 3). Each client's write request consists of a set of of s DPF keys $(k_1, \ldots, k_s)$, and the client sends one key to each server. If the database consists of L rows, the servers can expand these keys into s length-L vectors $(v_1, \ldots, v_s) \in (\mathbb{F}^L)^s$. The servers want to check that these vectors represent a write request that can update at most one row of the database. More precisely, the servers want to check that the vector $v = \sum_{j \in [s]} v_j \in \mathbb{F}^L$ has Hamming weight at most one.

Notice that this is *exactly* the setting of a zero-knowledge proof on secret-shared data: there are s verifiers (the servers), with each verifier holding an additive secret-sharing of a secret vector $v \in \mathbb{F}^n$, and there is a prover (the client) who knows v in its entirety. The verifiers want to check whether some predicate holds on v—in this case, the verifiers want to check that v has weight one.

More formally, the verifiers want to execute this check in such a way that the following properties hold:

- **Completeness.** If all parties are honest, and v has Hamming weight one at most, then the verifiers accept v.
- **Soundness.** If v has Hamming weight greater than one, and if all verifiers are honest, then the verifiers will reject v with high probability.
- **Zero knowledge.** If the client is honest, then any proper subset of actively malicious servers can simulate its view of the protocol execution (without knowledge of the vector v).

In Riposte, we can implement this write-request-checking functionality by directly applying any zero-knowledge proof on secret-shared data for the language of vectors of Hamming weight one. To do so, the client proves to the servers that the vector comprising its write request has Hamming weight one. The servers accept and process the client's write request if, and only if, this check passes.

In particular, we use the fully linear IOP for vectors of Hamming weight one of Theorem 2.4.12 and then compile it into a zero-knowledge proof on secret-shared data using the generic transformation of Theorem 3.2.1. The resulting proof system requires a constant number of rounds of interaction between the client and the servers, requires the servers to exchange a constant number of field elements to check each request, and has soundness error $O(1)/|\mathbb{F}|$, when used with an underlying field $\mathbb{F}$.

5.5.2 The s-server setting: New proofs on committed data

The s-server DPF construction of Section 5.4.4 outputs vectors of group elements $\mathbb{G}^n$, rather than field elements, so it is not immediately obvious how to apply our proofs on secret-shared to this setting. (In contrast, the two-server DPF construction of Section 5.4.3 natively outputs vectors of field elements.) With a bit more work, it is possible to apply our ideas on fully linear proof systems (Chapter 2) to this setting as well. The downside is that the s-server protocol requires public-key cryptographic operations, but since these are already required for the s-party DPF, this protocol just causes a small multiplicative increase in the servers' computational load.

In particular, when using Riposte in the s-server DPF, each server j receives a DPF key, which it expands to a vector $v^{(j)} \in \mathbb{G}^L$, where $\mathbb{G}$ is a group in which the DDH problem is hard. Then, if we take the sum of the j vectors $v^{(j)}, \ldots v^{(j)} \in \mathbb{G}^L$, we should get a vector v that is equal to the identity element in $\mathbb{G}$ everywhere except at a single location. That is, when $g \in \mathbb{G}$ is the fixed generator of the group, the vector v should have the following form, for some value $m \in \mathbb{Z}_q$:

$$v = \sum_{j \in [s]} v^{(j)} = \left(g^0, g^0, \ldots, g^0, g^m, g^0, \ldots, g^0\right) \quad \in \mathbb{G}^L.$$

In other words, we can think of the servers as holding additive shares *in the exponent* of a vector $(0, 0, \ldots, m, \ldots, 0, 0) \in \mathbb{Z}_q^L$ of Hamming weight one. With this view, we can now apply our proofs on secret-shared data to this setting.

In particular, we can essentially have the client and the servers run the fully linear proof system of Theorem 2.4.12 "in the exponent." This is possible because the servers can jointly compute linear functions of the exponent vector of v. To sketch how this works: the client and servers compile our fully linear proof system for vectors of Hamming weight one (Theorem 2.4.12) into a zero-knowledge proof on secret-shared data via the transformation

of Theorem 3.2.1.

Then, the client and servers execute this protocol, with the client playing the role of the prover and the servers playing the role of verifiers. When the servers need to make linear queries to their input vectors, they compute these queries answers "in the exponent" and then ask the client (prover) to prove—using a standard discrete-log-based zero-knowledge proof [69]—that these queries satisfy the fully linear PCP decision predicate. Since the discrete-log-based zero-knowledge proof has size constant in the dimension of the vector v, the overhead of this last step is asymptotically zero.

We sketch a slightly optimized verison of this protocol here. The protocol uses a fixed public generator $h \in \mathbb{G}$, whose discrete log base g should be unknown.

- The servers begin holding vectors $v^{(1)}, \ldots, v^{(s)} \in \mathbb{G}^L$, such that the vector $v = \sum_{j \in [x]} v^{(j)} \in \mathbb{G}^L$ should contain the identity element in $\mathbb{G}$ at all coordinates except one. Write $v = (v_1, \ldots, v_L) \in \mathbb{G}^L$. The client sends to the servers additive shares of blinding values ρ_1 and ρ_2, chosen at random from $\mathbb{Z}_q$.
- The servers choose a random exponent vector $r = (r_1, \ldots, r_L) \in \mathbb{Z}_q^L$ and send it to the client.
- The client and servers compute two test values $A_1, A_2 \in \mathbb{G}$ as:

$$A_1 \leftarrow \prod_{i \in [L]} v_i^{r_i} h^{\rho_1} \in \mathbb{G} \qquad \text{and} \qquad A_2 \leftarrow \prod_{i \in [L]} v_i^{r_i^2} h^{\rho_2} \in \mathbb{G},$$

  To compute A_1 and A_2, the servers can take the inner product of their shares of v with the client-provided vector r. By publishing the resulting values, the servers can recover A_1 and A_2.

  The values are commitments to the answers to the fully linear PCP queries that the verifier in the protocol of Theorem 2.4.12 asks.

- The client proves to the servers, using a standard discrete-log-based zero knowledge proof, knowledge of elements $\alpha, \rho_1, \rho_2 \in \mathbb{Z}_q$ such that

$$A_1 = g^{\alpha} h^{\rho_1} \in \mathbb{G} \qquad \text{and} \qquad A_2 = g^{\alpha^2} h^{\rho_2}.$$

  The client can execute this proof because it generated the values ρ_1 and ρ_2 and it can compute $\alpha = r_{i^*} m \in \mathbb{Z}_q$, where $i^* \in [L]$ is the index of the non-zero exponent in v and $m \in \mathbb{Z}_q$ is the value of the exponent vector of v at this point. The client here is proving

to the servers that the fully linear PCP query answers that it has committed to in A_1 and A_2 satisfy the fully linear PCP decision predicate.

Completeness and zero knowledge follow immediately. Soundness follows from the soundness of the underlying linear PCP and discrete-log-based zero-knowledge proof.

5.6 Experimental evaluation

To demonstrate that it is feasible to use Riposte as a platform for traffic-analysis-resistant anonymous messaging, we implemented the two-server variant of the system (Sections 5.4.3 and 5.5.1) This variant is relatively fast, since it relies exclusively on symmetric-key primitives (except as required for key exchange to establish secret channels). Our results *include the cost* of identifying and excluding malicious clients.

We wrote the prototype in the Go programming language and have published the source code online at https://bitbucket.org/henrycg/riposte/src/linear/. We used Amazon EC2 for our experiments. All of the experiments used c5n.4xlarge machines with 16 virtual CPU cores and 42 GB of RAM.

Our experimental network topology used two servers and two client nodes. In each of these experiments, the two client machines used many threads of execution to submit write requests to the servers as quickly as possible. In all experiments, clients proxied their (encrypted) write requests through the first server. We throttled the first server's network link to 100 Mbps and added 20 ms of latency to that link in either direction. We chose this network topology to limit the bandwidth between the servers to that of a fast WAN.

Error bars in the charts indicate the standard deviation of the throughput measurements.

5.6.1 Two-server protocol

We have fully implemented the two-server protocol so the throughput numbers listed here *include* the cost of detecting and rejecting malicious write requests.

The prototype used AES-128 in counter mode as the pseudo-random generator, TLS for link encryption, and NaCl Box to encrypt messages proxied through the first server. We use the Poly1305 [44], as the keyed hash function to implement the hashing step in the write-request-checking protocol, used in the audit protocol (see Remark 2.4.13).

Figure 5.2 shows how many client write requests our Riposte cluster can service per second as the number of 160-byte rows in the database table grows. For a database table of

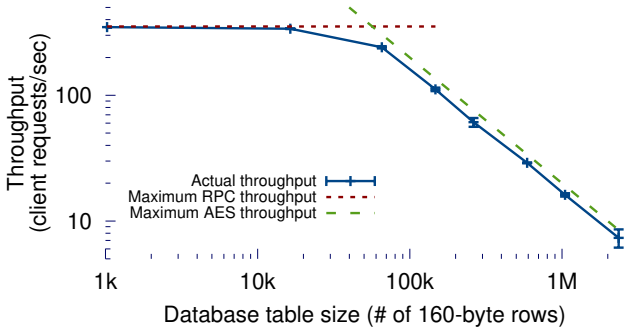


Figure 5.2: As the database table size grows, the throughput of our system is limited by the servers' AES throughput.

1024 rows, the system handles 349 write requests per second. At a table size of 65,536 rows, the system handles 241 requests per second. At a table size of 1,048,576 rows, the system handles 16 requests per second.

We chose the row length of 160 bytes because it was the smallest multiple of 32 bytes large enough to to contain a 140-byte Tweet. Throughput of the system depends only the total size of the table (number of rows $\times$ row length), so larger row lengths might be preferable for other applications. For example, an anonymous email system using Riposte with 4096-byte rows could handle 16 requests per second at a table size of 40,960 rows.

An upper bound on the performance of the system is the speed of the pseudo-random generator used to stretch out the DPF keys to the length of the database table. The dashed line in Figure 5.2 indicates this upper bound. To compute this upper bound, we first find the raw AES throughput (15 GB/s) using the `openssl speed` utility. Then, we divide by five to account for the fact that our implementation requires performing five AES byte operations per byte in the database table. With additional optimizations, we could get this cost down to three AES byte operations per byte in the table (one for TLS, one to expand the DPF key, and one for the proof of correctness), but getting beyond that would require new ideas. At very small table sizes, the speed at which the server can handle RPC connections with the clients over TLS limits the overall throughput to roughly 350 requests per second. Since our write-request-checking protocol of Section 5.5.1 requires the client to make three RPC calls per write request, we could conceivably get a factor of 3$\times$ performance improvement by converting these protocols into non-interactive ones.

Figure 5.3 demonstrates how the request throughput varies as the width of the table

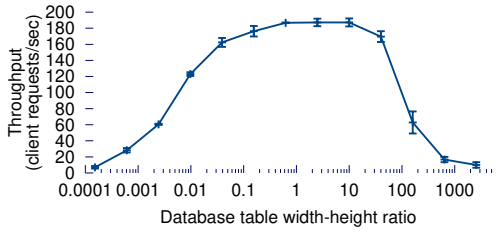


Figure 5.3: Use of bandwidth-efficient DPFs gives a 25× speed-up over the naïve constructions, in which a client's request is as large as the database.

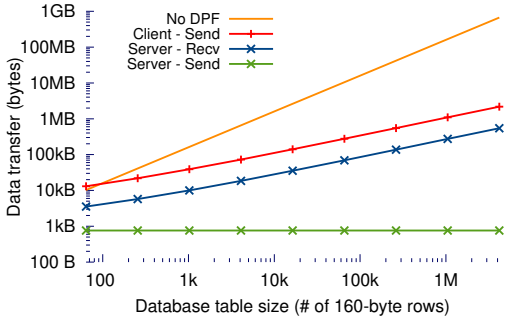


Figure 5.4: The total client and server data transfer scales sub-linearly with the size of the database.

changes, while the number of bytes in the table is held constant at 10 MB. This figure demonstrates the performance advantage of using a bandwidth-efficient $O(\sqrt{L})$ DPF (Section 5.4) over the naïve DPF (Section 5.3.1). Using a DPF with optimal table size yields a throughput roughly 25× higher than the naïve construction that involves sending dimension-L vector to each server.

Figure 5.4 indicates the total number of bytes transferred by one of the database servers and by the audit server while processing a single client write request. The line at the top of the chart indicates the number of bytes a client would need to send for a single write request if we did not use bandwidth-efficient DPFs (i.e., the top line indicates the size of the database table). As the figure demonstrates, the total data transfer in a Riposte cluster scales *sub-linearly* with the database size. When the database table is 671 MB in size, the client transfers only a total of 2.18 MB to process a write request. Furthermore, the database servers transfer only a *constant* number of bytes (760 bytes) to check the correctness of the client's write request, independent of the size of the database table.

5.6.2 Discussion: Whistleblowing with million-user anonymity sets

Whistleblowers, political activists, or others discussing sensitive or controversial issues might benefit from an anonymous microblogging service. A whistleblower, for example, might want to anonymously blog about an instance of bureaucratic corruption in her organization. The utility of such a system depends on the size of the anonymity set it would provide: if a whistleblower is only anonymous amongst a group of ten people, it would be easy for the whistleblower's employer to retaliate against *everyone* in the anonymity set. Mounting this

"punish-them-all" attack does not require breaking the anonymity system itself, since the anonymity set is public. As the anonymity set size grows, however, the feasibility of the "punish-them-all" attack quickly tends to zero. At an anonymity set size of 1,000,000 clients, mounting an "punish-them-all" attack would be prohibitively expensive in most situations.

Riposte can handle such large anonymity sets as long as (1) clients are willing to tolerate hours of messaging latency, and (2) only a small fraction of clients writes into the database in each time epoch. Both of these requirements are satisfied in the whistleblowing scenario. First, whistleblowers might not care if the system delays their posts by a few hours. Second, the vast majority of users of a microblogging service (especially in the whistleblowing context) are more likely to *read* posts than write them. To get very large anonymity sets, maintainers of an anonymous microblogging service could take advantage of the large set of "read-only" users to provide anonymity for the relatively small number of "read-write" users.

The client application for such a microblogging service would enable read-write users to generate and submit Riposte write requests to a Riposte cluster running the microblogging service. However, the client application would also allow read-only users to submit an "empty" write request to the Riposte cluster that would always write a random message into the first row of the Riposte database. From the perspective of the servers, a read-only client would be indistinguishable from a read-write client. By leveraging read-only users in this way, we can increase the size of the anonymity set without needing to increase the size of the database table.

To demonstrate that Riposte can support very large anonymity set sizes—albeit with high latency—we configured a cluster of Riposte servers with a 65,536-row database table and left it running for just over seven hours. In that period, the system processed a total of 6,162,647 write requests at an average rate of 240.9 requests per second. Using the techniques in Section 5.3.2, a table of this size could handle 0.3% of users writing at a collision rate of under 5%. Thus, to get an anonymity set of roughly 1,000,000 users with a two-server Riposte cluster and a database table of size $65,536$, the time epoch must be just over an hour long.

As of 2019, Twitter statistics indicate an average throughput of 8,700 Tweets per second [168]. At a table size of one million messages, our Riposte cluster's end-to-end throughput is 16.4 write requests per second (Figure 5.2). To handle the same volume of Tweets as Twitter does with anonymity set sizes on the order of hundreds of thousands of clients, we would need to increase the computing power of our cluster by "only" a factor of $\approx 530\times$. (We

assume here that scaling the number of machines by a factor of k increases our throughput by a factor of k. This assumption is reasonable given our workload, since the processing of write requests is an embarrassingly parallel task.) Since we are using only two servers now, we would need roughly 1,060 servers, split into two non-colluding data centers, to handle the same volume of traffic as Twitter.

5.7 Related Work

Anonymity systems fall into one of two general categories: systems that provide low-latency communication and systems that protect against traffic analysis attacks by a global network adversary.

Aqua [197], Crowds [235], LAP [166], ShadowWalker [211], Tarzan [129], and Tor [112] belong to the first category of systems: they provide an anonymous proxy for real-time Web browsing, but they do not protect against an adversary who controls the network, many of the clients, and some of the nodes on a victim's path through the network. Even providing a formal definition of anonymity for low-latency systems is challenging [175] and such definitions typically do not capture the need to protect against timing attacks.

Even so, it would be possible to combine Tor (or another low-latency anonymizing proxy) and Riposte to build a "best of both" anonymity system: clients would submit their write requests to the Riposte servers via the Tor network. In this configuration, even if *all* of the Riposte servers colluded, they could not learn which user wrote which message without also breaking the anonymity of the Tor network.

David Chaum's "cascade" mix networks were one of the first systems devised with the specific goal of defending against traffic-analysis attacks [78]. Since then, there have been a number of mix-net-style systems proposed, many of which explicitly weaken their protections against a near omni-present adversary [255] to improve prospects for practical usability (i.e., for email traffic) [106]. In contrast, Riposte attempts to provide very strong anonymity guarantees at the price of usability for interactive applications.

E-voting systems (also called "verifiable shuffles") achieve the sort of privacy properties that Riposte offers, and some systems even provide stronger voting-specific guarantees (receipt-freeness, proportionality, etc.), though most e-voting systems cannot provide the forward security property that Riposte offers (Section 5.3.3) [2, 88, 130, 155, 158, 220, 232].

In a typical e-voting system, voters submit their encrypted ballots to a few trustees, who

collectively shuffle and decrypt them. While it is possible to repurpose e-voting systems for anonymous messaging, they typically require expensive zero-knowledge proofs or are inefficient when message sizes are large. Mix-nets that do not use zero-knowledge proofs of correctness typically do not provide privacy in the face of active attacks by a subset of the mix servers.

For example, the verifiable shuffle protocol of Bayer and Groth [23] is one of the most efficient in the literature. Their shuffle implementation, when used with an anonymity set of size N, requires $16N$ group exponentiations per server and data transfer $O(N)$. In addition, messages must be small enough to be encoded in single group elements (a few hundred bytes at most). In contrast, our protocol requires $O(L)$ AES operations and data transfer $O(\sqrt{L})$, where L is the size of the database table. When messages are short and when the writer/reader ratio is high, the Bayer-Groth mix may be faster than our system. In contrast, when messages are long and when the writer/reader ratio is low (i.e., $L \ll O(N)$), our system is faster.

Chaum's Dining Cryptographers network (DC-net) is an information-theoretically secure anonymous broadcast channel [77]. A DC-net provides the same strong anonymity properties as Riposte does, but it requires every user of a DC-net to participate in every run of the protocol. As the number of users grows, this quickly becomes impractical.

The Dissent [274] system introduced the idea of using partially trusted servers to make DC-nets practical in distributed networks. Dissent requires weaker trust assumptions than our three-server protocol does but it requires clients to send $O(L)$ bits to each server per time epoch (compared with our $O(\sqrt{L})$). Also, excluding *a single* disruptor in a 1,000-client deployment takes over an hour. In contrast, Riposte can excludes disruptors as fast as it processes write requests (tens to hundreds per second, depending on the database size). Recent work [96] uses zero-knowledge techniques to speed up disruption resistance in Dissent (building on ideas of Golle and Juels [153]). Unfortunately, these techniques limit the system's end to end-throughput end-to-end throughput to 30 KB/s, compared with Riposte's 450+ MB/s.

Herbivore scales DC-nets by dividing users into many small anonymity sets [141]. Riposte creates a single large anonymity set, and thus enables every client to be anonymous amongst the *entire set* of honest clients.

Our DPF constructions make extensive use of prior work on private information retrieval (PIR) [84, 86, 131, 138]. Recent work demonstrates that it is possible to make theoretical PIR

fast enough for practical use [110, 111, 143]. Function secret sharing [62, 63] generalizes DPFs to allow sharing of more sophisticated functions (rather than just point functions). This more powerful primitive may prove useful for PIR and anonymous messaging applications in the future.

Gertner et al.[136] consider *symmetric* PIR protocols, in which the servers prevent dishonest clients from learning about more than a single row of the database per query. The problem that Gertner et al. consider is, in a way, the dual of the problem we address in Section 5.5, though their techniques do not appear to apply directly in our setting.

Ostrovsky and Shoup first proposed using PIR protocol as the basis for writing into a database shared across a set of servers [223]. However, Ostrovsky and Shoup considered only the case of a single honest client, who uses the untrusted database servers for private storage. Since *many mutually distrustful clients* use a single Riposte cluster, our protocol must also handle malicious clients.

Pynchon Gate [239] builds a private point-to-point messaging system from mix-nets and PIR. Clients anonymously upload messages to email servers using a traditional mix-net and download messages from the email servers using a PIR protocol. Riposte could replace the mix-nets used in the Pynchon Gate system: clients could anonymously write their messages into the database using Riposte and could privately read incoming messages using PIR.

Subsequent related work. Since the publication of the initial version of this work [94], there has been much progress on large-scale systems for private and anonymous messaging. One line of work, including the Vuvuzela [262], Alpenhorn [196], Stadium [260], and Karaoke [194] systems, aim to build large-scale messaging systems that hide communications metadata. These systems do *not* allow anonymous broadcast messaging, as we do in Riposte, but target the problem of point-to-point communication between mutually trusting users. In this very well-motivated setting, these works construct systems that scale to many millions of users with surprisingly low latency—minutes or even seconds. One way these systems achieve such performance is by providing a differential-privacy-style [117] notion of security, which is in some sense a relaxation of the more traditional cryptographic notions of metadata privacy.

The Pung system [10, 11] proposes a radically different to point-to-point private messaging, based on single-server private-information retrieval protocols [190]. Pung has the benefit of not requiring users to make *any* trust or non-collusion assumptions about the system's servers, which is undoubtedly desirable in practice. This removal of trust assumptions comes

at some performance cost: today's single-server PIR schemes make heavy use of public-key cryptographic primitives, so Pung must as well. In addition, Pung does not support anonymous broadcast messaging, so it may be less suitable for whistleblowing applications.

Another approach has been to develop new techniques for scaling up classic anonymity systems based on Chaum's mix-networks [78]. The Atom system [191] builds a large distributed mix-network-based anonymity system, inspired by prior theoretical work of Rackoff and Simon [233]. The goal of these systems is to have each server process a small fraction of the total traffic through the system, while allowing each user to benefit from an anonymity set that includes all users of the system. The XRD system [193] also uses distributed mix networks, but uses it to build a metadata-private point-to-point messaging system (rather than anonymous broadcast). Loopix [229] is a distributed free-route mix-net, designed for low-latency asynchronous messaging. While Loopix does not provide formal (i.e., cryptographic) security guarantees, it has the potential to scale to very large network sizes. MCMix [5] implements the ideal functionality of a private communication system directly using general-purpose multi-party computation techniques.

Riffle [192] constructs a fast hybrid mix-net architecture that uses an expensive verifiable shuffle to set up a fast "bulk transfer" phase (e.g., for sending large files). Yodel [195] similarly develops a hybrid mix architecture using a different set of techniques with the goal of getting latency low enough for real-time metadata-private voice calls.

5.8 Conclusion

We have presented Riposte, a new system for anonymous messaging. To the best of our knowledge, Riposte was the first system that could simultaneously (1) thwart traffic analysis attacks, (2) prevent malicious clients from anonymously disrupting the system, and (3) enable million-client anonymity set sizes. We achieve these goals through novel application of private information retrieval and secure multiparty computation techniques. We have demonstrated Riposte's practicality by implementing it and evaluating it with anonymity sets of over six million nodes. With the design and implementation of Riposte, we have demonstrated that cryptographic techniques can bring traffic-analysis-resistant anonymous microblogging and whistleblowing closer to practice at Internet scale.

Part III

Conclusion

When you give your sensitive information to a web service today, you have no control over how that service will use your data. Will it store your data safely? Will it resell your data for profit? Will it lose your data in a breach? You, as the user whose information is at risk, have no idea. This is a miserable state of affairs. We tolerate it only because we have no other option.

In this dissertation, we present a way forward. We show that it is possible to design large-scale systems that protect the privacy of sensitive user data by *splitting trust* among multiple entities. As long as any one of these entities behaves honestly and stores its secrets safely, users get strong and precise guarantees of security and privacy.

By constructing two systems—one for the private computation of aggregate statistics and one for anonymous messaging—we have proven that splitting trust is possible in principle. By deploying the first of these systems in the Firefox web browser, we have demonstrated that splitting trust is possible in practice as well. Application of cryptographic techniques, such as our new zero-knowledge proofs on distributed data, is at the core of our approach.

Our computer systems must do a better job of keeping us in control of our personal data. Splitting trust is one promising way to achieve this goal. Whatever techniques we use, we should expect more from our computer systems in terms of the privacy properties they provide. I am optimistic that the next generation of computer systems will do a better job of serving our interests and protecting our privacy.

Bibliography

[1] Scott Aaronson and Avi Wigderson. "Algebrization: a new barrier in complexity theory." *ACM Transactions on Computation Theory* 1.1 (2009).

[2] Ben Adida. "Helios: Web-based open-audit voting." *USENIX Security Symposium*. 2008.

[3] Ben Adida and Douglas Wikström. "How to shuffle in public." *IACR Theory of Cryptography Conference*. 2007.

[4] Miklós Ajtai. "Generating hard instances of lattice problems." *ACM Symposium on Theory of Computing*. 1996.

[5] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. "MCMix: anonymous messaging via secure multiparty computation." *USENIX Security Symposium*. 2017.

[6] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. "SoK: the evolution of Sybil defense via social networks." *IEEE Symposium on Security and Privacy*. 2013.

[7] American Psychological Association. *Beck Depression Inventory*. http://www.apa.org/pi/about/publications/caregivers/practice-settings/assessment/tools/beck-depression.aspx. accessed 15 September 2016.

[8] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: lightweight sublinear arguments without a trusted setup." *ACM Conference on Computer and Communications Security*. 2017.

[9] C.-C. Yao Andrew. "Some complexity questions related to distributed computing." *ACM Symposium on Theory of Computing*. 1979.

[10] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. "PIR with compressed queries and amortized query processing." *IEEE Symposium on Security and Privacy*. 2018.

[11] Sebastian Angel and Srinath Setty. "Unobservable communication over fully untrusted infrastructure." *USENIX Symposium on Operating Systems Design and Implementation*. 2016.

[12] Benny Applebaum, Haakon Ringberg, Michael J Freedman, Matthew Caesar, and Jennifer Rexford. "Collaborative, privacy-preserving data aggregation at scale." *Privacy Enhancing Technologies Symposium*. 2010.

[13] Branden Archer and Eric W. Weisstein. *Lagrange Interpolating Polynomial*. http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html. accessed 16 September 2016.

[14] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof verification and the hardness of approximation problems." *Journal of the ACM* 45.3 (1998).

[15] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: a new characterization of NP." *Journal of the ACM* 45.1 (1998).

[16] László Babai. "Trading group theory for randomness." *ACM Symposium on Theory of Computing*. 1985.

[17] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking computations in polylogarithmic time." *ACM Symposium on Theory of Computing*. 1991.

[18] László Babai and Shlomo Moran. "Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes." *Journal of Computer and System Sciences* 36.2 (1988).

[19] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. "ADSNARK: nearly practical and privacy-preserving proofs on authenticated data." *IEEE Symposium on Security and Privacy*. 2015.

[20] Abhishek Banerjee and Chris Peikert. "New and improved key-homomorphic pseudorandom functions." *CRYPTO*. 2014.

[21] Raef Bassily and Adam Smith. "Local, private, efficient protocols for succinct histograms." *ACM Symposium on Theory of Computing*. 2015.

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

[23] Stephanie Bayer and Jens Groth. "Efficient zero-knowledge argument for correctness of a shuffle." *EUROCRYPT*. 2012.

[24] Donald Beaver. "Efficient multiparty protocols using circuit randomization." *CRYPTO*. 1991.

[25] Donald Beaver. "Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority." *Journal of Cryptology* 4.2 (1991).

[26] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The round complexity of secure protocols." *ACM Symposium on Theory of Computing*. 1990.

[27] Pete Bell. "Cable is main form of broadband access in North America." *TeleGeography* (Nov. 8, 2018). https://blog.telegeography.com/cable-is-main-form-of-broadband-access-in-north-america, accessed 18 October 2019.

[28] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. "Efficient garbling from a fixed-key blockcipher." *IEEE Symposium on Security and Privacy*. 2013.

[29] Mihir Bellare and Phillip Rogaway. "Random oracles are practical: a paradigm for designing efficient protocols." *ACM Conference on Computer and Communications Security*. 1993.

[30] Assaf Ben-David, Noam Nisan, and Benny Pinkas. "FairplayMP: a system for secure multi-party computation." *ACM Conference on Computer and Communications Security*. 2008.

[31] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. "Everything provable is provable in zero-knowledge." *CRYPTO*. 1988.

[32] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. "Multi-prover interactive proofs: how to remove intractability assumptions." *ACM Symposium on Theory of Computing*. 1988.

[33] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation." *ACM Symposium on Theory of Computing*. 1988.

[34] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable zero knowledge with no trusted setup." *CRYPTO*. 2019.

[35] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "On probabilistic checking in perfect zero knowledge." *Electronic Colloquium on Computational Complexity*. 2016.

[36] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. "Interactive oracle proofs with constant rate and query complexity." *International Colloquium on Automata, Languages and Programming*. 2017.

[37] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. "SNARKs for C: verifying program executions succinctly and in zero knowledge." *CRYPTO*. 2013.

[38] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: transparent succinct arguments for R1CS." *EUROCRYPT*. 2019.

[39] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive oracle proofs." *IACR Theory of Cryptography Conference*. 2016.

[40] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable zero knowledge via cycles of elliptic curves." *CRYPTO*. 2014.

[41] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct non-interactive zero knowledge for a von Neumann architecture." *USENIX Security Symposium*. 2014.

[42] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. "Robust PCPs of proximity, shorter PCPs, and applications to coding." *SIAM Journal on Computing* 36.4 (2006).

[43] Katrin Bennhold. "In Britain, guidelines for spying on lawyers and clients." *New York Times* (Nov. 7, 2014).

[44] Daniel J. Bernstein. "The Poly1305-AES message-authentication code." *Fast Software Encryption*. 2005.

[45] Oliver Berthold and Heinrich Langos. "Dummy traffic against long term intersection attacks." *Workshop on Privacy Enhancing Technologies*. 2002.

[46] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again." *Innovations in Theoretical Computer Science*. 2012.

[47] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. "Succinct non-interactive arguments via linear interactive proofs." *IACR Theory of Cryptography Conference*. 2013.

[48] George Robert Blakley. "Safeguarding cryptographic keys." *Proceedings of the National Computer Conference*. Vol. 48. 313. 1979.

[49] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. "Practical privacy: the SuLQ framework." *ACM SIGMOD Symposium on Principles of Database Systems*. 2005.

[50] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-interactive zero-knowledge and its applications." *ACM Symposium on Theory of Computing*. 1988.

[51] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-interactive zero-knowledge and its applications (extended abstract)." *ACM Symposium on Theory of Computing*. 1988.

[52] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. "Multiparty computation goes live." *Financial Cryptography*. 2000.

[53] Dan Boneh. "The decision Diffie-Hellman problem." *Algorithmic Number Theory*. Ed. by Joe P. Buhler. Vol. 1423. Lecture Notes in Computer Science. 1998.

[54] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Zero-knowledge proofs on secret-shared data via fully linear pcps." *CRYPTO*. 2019.

[55] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. "Lattice-based SNARGs and their application to more efficient obfuscation." *EUROCRYPT*. 2017.

[56] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. "Quasi-optimal SNARGs via linear multi-prover interactive proofs." *EUROCRYPT*. 2018.

[57] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. "Key homomorphic PRFs and their applications." *CRYPTO*. 2013.

[58] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting." *EUROCRYPT*. 2016.

[59] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. "Linear-time zero-knowledge proofs for arithmetic circuit satisfiability." *ASIACRYPT*. 2017.

[60] Jonathan Bootle and Jens Groth. "Efficient batch zero-knowledge arguments for low degree polynomials." *IACR International Conference on Practice and Theory in Public Key Cryptography*. 2018.

[61] Jurjen Bos and Bert den Boer. "Detection of disrupters in the DC protocol." *EUROCRYPT*. 1989.

[62] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function secret sharing." *EUROCRYPT*. 2015.

[63] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function secret sharing: improvements and extensions." *ACM Conference on Computer and Communications Security*. 2016.

[64] Justin Brickell and Vitaly Shmatikov. "Efficient anonymity-preserving data collection." *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2006.

[65] Anne Broadbent and Alain Tapp. "Information-theoretic security without an honest majority." *ASIACRYPT*. 2007.

[66] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: efficient range proofs for confidential transactions." *IEEE Symposium on Security and Privacy*. 2018.

[67] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK compilers" (2019).

[68] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. "SEPIA: privacy-preserving aggregation of multi-domain network events and statistics." *USENIX Security Symposium* (2010).

[69] Jan Camenisch and Markus Stadler. *Proof systems for general statements about discrete logarithms*. Tech. rep. 260. Dept. of Computer Science, ETH Zürich, 1997.

[70] Dave Camp. *Firefox Now Available with Enhanced Tracking Protection by Default Plus Updates to Facebook Container, Firefox Monitor and Lockwise*. The Mozilla Blog. https://blog.mozilla.org/blog/2019/06/04/firefox-now-available-with-enhanced-tracking-protection-by-default/, accessed 23 October 2019. June 4, 2019.

[71] Ran Canetti, Shai Halevi, and Jonathan Katz. "A forward-secure public-key encryption scheme." *EUROCRYPT*. 2003.

[72] Claude Castelluccia, Einar Mykletun, and Gene Tsudik. "Efficient aggregation of encrypted data in wireless sensor networks." *MobiQuitous*. IEEE. 2005.

[73] Dario Catalano and Dario Fiore. "Vector commitments and their applications." *IACR International Conference on Practice and Theory in Public Key Cryptography*. 2013.

[74] Ariana Eunjung Cha and Sam Diaz. "Advocates sue Yahoo in Chinese torture case." *Washington Post* (Apr. 19, 2007). http://www.washingtonpost.com/wp-dyn/content/article/2007/04/18/AR2007041802510.html, accessed 21 October 2019.

[75] Amit Chakrabarti. *CS49: Data Stream Algorithms, Lecture Notes, Fall 2011*. accessed 12 September 2016. www.cs.dartmouth.edu/~ac/Teach/CS49-Fall11/Notes/lecnotes.pdf. Oct. 2014.

[76] Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. "Annotations in data streams." *ACM Transactions on Algorithms* 11.1 (2014).

[77] David Chaum. "The Dining Cryptographers Problem: unconditional sender and recipient untraceability." *Journal of Cryptology* 1.1 (1988).

[78] David L. Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms." *Communications of the ACM* 24.2 (1981).

[79] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty unconditionally secure protocols." *ACM Symposium on Theory of Computing*. 1988.

[80] Ruichuan Chen, Istemi Ekin Akkus, and Paul Francis. "SplitX: high-performance private analytics." *SIGCOMM* (2013).

[81] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johanes Gehrke. "Towards statistical queries over distributed private user data." *USENIX Symposium on Networked Systems Design and Implementation*. 2012.

[82] Robert T. Chien and W. D. Frazer. "An application of coding theory to document retrieval." *IEEE Transactions on Information Theory* 12.2 (1966).

[83] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. "A zero knowledge sumcheck and its applications." *arXiv preprint arXiv:1704.02086* (2017).

[84] Benny Chor and Niv Gilboa. "Computationally private information retrieval." *ACM Symposium on Theory of Computing*. ACM. 1997.

[85] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private information retrieval." *IEEE Symposium on Foundations of Computer Science*. 1995.

[86] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private information retrieval." *Journal of the ACM* 45.6 (1998).

[87] *Chromium Source Code*. https://chromium.googlesource.com/chromium/src/+/master/tools/metrics/rappor/rappor.xml. accessed 15 September 2016. Sept. 15, 2016.

[88] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. *Civitas: A secure voting system*. Tech. rep. TR 2007-2081. Cornell University, May 2007.

[89] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. "Characterization of secure multiparty computation without broadcast." *Journal of Cryptology* 31.2 (2018).

[90] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. "Practical verified computation with streaming interactive proofs." *Innovations in Theoretical Computer Science*. 2012.

[91] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." *Journal of Algorithms* 55.1 (2005).

[92] Graham Cormode, Justin Thaler, and Ke Yi. "Verifying computations with streaming interactive proofs." *International Conference on Very Large Data Bases*. 2011.

[93] Henry Corrigan-Gibbs and Dan Boneh. "Prio: private, robust, and scalable computation of aggregate statistics." *USENIX Symposium on Networked Systems Design and Implementation*. 2017.

[94] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. "Riposte: an anonymous messaging system handling millions of users." *IEEE Symposium on Security and Privacy*. 2015.

[95] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: accountable anonymous group messaging." *ACM Conference on Computer and Communications Security*. 2010.

[96] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. "Proactively accountable anonymous messaging in Verdict." *USENIX Security Symposium*. 2013.

[97] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. "Geppetto: versatile verifiable computation." *IEEE Symposium on Security and Privacy*. 2015.

[98] CPP. *California Psychological Inventory*. https://www.cpp.com/products/cpi/index.aspx. accessed 15 September 2016.

[99] Ronald Cramer and Victor Shoup. "A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack." *CRYPTO*. 1998.

[100] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. "Implementing AES via an actively/covertly secure dishonest-majority MPC protocol." *SCN*. 2012.

[101] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. "Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits." *European Symposium on Research in Computer Security*. 2013.

[102] Ivan Damgård, Ji Luo, Sabine Oechsner, Peter Scholl, and Mark Simkin. "Compact zero-knowledge proofs of small Hamming weight." *IACR International Conference on Practice and Theory in Public Key Cryptography*. 2018.

[103] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty computation from somewhat homomorphic encryption." *CRYPTO*. 2012.

[104] George Danezis. "Better anonymous communications." PhD thesis. University of Cambridge, 2004.

[105] George Danezis and Claudia Diaz. *A survey of anonymous communication channels*. Tech. rep. MSR-TR-2008-35. Microsoft Research, 2008.

[106] George Danezis, Roger Dingledine, and Nick Mathewson. "Mixminion: design of a type III anonymous remailer protocol." *IEEE Symposium on Security and Privacy*. 2003.

[107] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. "Smart meter aggregation via secret-sharing." *Workshop on Smart Energy Grid Security*. ACM. 2013.

[108] George Danezis and Andrei Serjantov. "Statistical disclosure or intersection attacks on anonymity systems." *Information Hiding*. 2004.

[109] DEDIS Research Lab at EPFL. *Advanced crypto library for the Go language*. https://github.com/dedis/crypto.

[110] Daniel Demmler, Amir Herzberg, and Thomas Schneider. "RAID-PIR: practical multi-server PIR." *Workshop on Privacy in the Electronic Society*. 2014.

[111] Casey Devet and Ian Goldberg. "The best of both worlds: combining information-theoretic and computational pir for communication efficiency." *Privacy Enhancing Technologies Symposium*. 2014.

[112] Roger Dingledine, Nick Mathewson, and Paul Syverson. "Tor: the second-generation onion router." *USENIX Security Symposium*. 2004.

[113] John R. Douceur. "The Sybil Attack." *International Workshop on Peer-to-Peer Systems*. 2002.

[114] Yitao Duan, John Canny, and Justin Zhan. "P4P: practical large-scale privacy-preserving distributed computation robust against malicious users." *USENIX Security Symposium*. 2010.

[115] Cynthia Dwork. "Differential privacy." *International Colloquium on Automata, Languages and Programming*. 2006.

[116] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. "Our data, ourselves: privacy via distributed noise generation." *EUROCRYPT*. 2006.

[117] Cynthia Dwork and Aaron Roth. "The algorithmic foundations of differential privacy." 9.3–4 (2014).

[118] Matthew Edman and Bülent Yener. "On anonymity in an electronic society: a survey of anonymous communication systems." *ACM Computing Surveys* 42.1 (2009).

[119] Tariq Elahi, George Danezis, and Ian Goldberg. "PrivEx: private collection of traffic statistics for anonymous communication networks." *ACM Conference on Computer and Communications Security*. 2014.

[120] Steven Englehardt. *Next steps in privacy-preserving Telemetry with Prio*. Mozilla Security Blog. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/, accessed 23 October 2019. June 6, 2019.

[121] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: randomized aggregatable privacy-preserving ordinal response." *ACM Conference on Computer and Communications Security*. 2014.

[122] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. "Express: lowering the cost of metadata-hiding communication with cryptographic privacy." *arXiv preprint arXiv:1911.09215* (2019).

[123] Facebook. *Company Info | Facebook Newsroom*. https://newsroom.fb.com/company-info/, accessed 18 October 2019. Oct. 2019.

[124] Giulia Fanti, Vasyl Pihur, and Úlfar Erlingsson. "Building a RAPPOR with the unknown: privacy-preserving learning of associations and data dictionaries." *Proceedings on Privacy Enhancing Technologies*. Vol. 2016.

[125] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. "Approximating clique is almost NP-complete." *IEEE Symposium on Foundations of Computer Science*. 1991.

[126] Amos Fiat and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems." *CRYPTO*. 1986.

[127] *FLINT: Fast Library for Number Theory*. http://www.flintlib.org/.

[128] Lance Fortnow, John Rompel, and Michael Sipser. "On the power of multi-prover interactive protocols." *Theoretical Computer Science* 134.2 (1994).

[129] Michael J. Freedman and Robert Morris. "Tarzan: a peer-to-peer anonymizing network layer." *ACM Conference on Computer and Communications Security*. ACM. 2002.

[130] Jun Furukawa. "Efficient, verifiable shuffle decryption and its requirement of unlinkability." *IACR International Conference on Practice and Theory in Public Key Cryptography*. 2004.

[131] William Gasarch. "A survey on private information retrieval." *Bulletin of the EATCS*. 2004.

[132] Barton Gellman and Ashkan Soltani. "NSA infiltrates links to Yahoo, Google data centers worldwide, Snowden documents say." *Washington Post* (Oct. 30, 2013).

[133] Barton Gellman, Julie Tate, and Ashkan Soltani. "In NSA-intercepted data, those not targeted far outnumber the foreigners who are." *Washington Post* (July 5, 2014).

[134] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic span programs and succinct NIZKs without PCPs." *CRYPTO*. 2013.

[135] Anthony B. Gerard. *Parent-Child Relationship Inventory*. http://www.wpspublish.com/store/p/2898/parent-child-relationship-inventory-pcri. accessed 15 September 2016.

[136] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. "Protecting data privacy in private information retrieval schemes." *ACM Symposium on Theory of Computing*. 1998.

[137] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. "Protecting data privacy in private information retrieval schemes." *Journal of Computer and System Sciences* 60.3 (2000).

[138] Niv Gilboa and Yuval Ishai. "Distributed point functions and their applications." *EUROCRYPT*. 2014.

[139] James Glanz, Jeff Larson, and Andrew W. Lehren. "Spy agencies tap data streaming from phone apps." *New York Times* (Jan. 27, 2014). accessed 20 September 2016.

[140] Jeff Glaze. "Epic Systems draws on literature greats for its next expansion." *Wisconsin State Journal* (Jan. 6, 2015). https://madison.com/news/local/govt-and-politics/epic-systems-draws-on-literature-greats-for-its-next-expansion/article_4d1cf67c-2abf-5cfd-8ce1-2da60ed84194.html, accessed 18 October 2019.

[141] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. *Herbivore: A scalable and efficient protocol for anonymous communication*. Tech. rep. TR2003-1890. Cornell University, 2003.

[142] Vindu Goel. "Government push for Yahoo's user data set stage for broad surveillance." *New York Times* (Sept. 7, 2014).

[143] Ian Goldberg. "Improving the robustness of private information retrieval." *IEEE Symposium on Security and Privacy*. 2007.

[144] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[145] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Collision-free hashing from lattice problems" (1996).

[146] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to play any mental game." *ACM Symposium on Theory of Computing*. 1987.

[147] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems." *Journal of the ACM* 38.3 (1991).

[148] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for Muggles." *ACM Symposium on Theory of Computing*. 2008.

[149] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating computation: interactive proofs for Muggles." *Journal of the ACM* 62.4 (2015).

[150] Shafi Goldwasser and Yehuda Lindell. "Secure multi-party computation without agreement." *Journal of Cryptology* 18.3 (2005).

[151] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The knowledge complexity of interactive proof systems." *SIAM Journal on Computing* 18.1 (1989).

[152] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. "A digital signature scheme secure against adaptive chosen-message attacks." *SIAM Journal on Computing* 17.2 (1988).

[153] Philippe Golle and Ari Juels. "Dining Cryptographers revisited." *EUROCRYPT*. 2004.

[154] Andy Greenberg. "Apple's 'differential privacy' is about collecting your data—but not your data." *Wired* (June 13, 2016). accessed 21 September 2016.

[155] Jens Groth. "A verifiable secret shuffle of homomorphic encryptions." *Journal of Cryptology* 23.4 (2010).

[156] Jens Groth. "Short pairing-based non-interactive zero-knowledge arguments." *ASIACRYPT*. 2010.

[157] Jens Groth. "On the size of pairing-based non-interactive arguments." *EUROCRYPT*. 2016.

[158] Jens Groth and Steve Lu. "Verifiable shuffle of large size ciphertexts." *IACR International Conference on Practice and Theory in Public Key Cryptography*. 2007.

[159] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. "Fast garbling of circuits under standard assumptions." *ACM Conference on Computer and Communications Security*. 2015.

[160] Sudipto Guha and Andrew McGregor. "Stream order and order statistics: quantile estimation in random-order streams." *SIAM Journal on Computing* 38.5 (2009).

[161] Tom Gur and Ron D. Rothblum. "A hierarchy theorem for interactive proofs of proximity." *Innovations in Theoretical Computer Science*. 2017.

[162] Robert Helmer, Anthony Miyaguchi, and Eric Rescorla. *Testing Privacy-Preserving Telemetry with Prio*. Mozilla Hacks Blog. https://hacks.mozilla.org/2018/10/testing-privacy-preserving-telemetry-with-prio/, accessed 23 October 2019. Oct. 29, 2018.

[163] Alex Hern. "Uber employees 'spied on ex-partners, politicians and beyoncé'" (Dec. 13, 2016). https://www.theguardian.com/technology/2016/dec/13/uber-employees-spying-ex-partners-politicians-beyonce, accessed 21 October 2019.

[164] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. *Every step you fake: a comparative analysis of fitness tracker privacy and security*. Tech. rep. accessed 16 September 2016. Open Effect, 2016.

[165] Susan Hohenberger, Steven Myers, Rafael Pass, and abhi shelat. "ANONIZE: a large-scale anonymous survey system." *IEEE Symposium on Security and Privacy*. 2014.

[166] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Adrian Perrig, Akira Yamada, Samuel C. Nelson, Marco Gruteser, and Wei Meng. "LAP: lightweight anonymity and privacy." *IEEE Symposium on Security and Privacy*. 2012.

[167] Russell Impagliazzo and Moni Naor. "Efficient cryptographic schemes provably as secure as subset sum." *Journal of Cryptology* 9.4 (1996).

[168] Internet Live Stats. https://www.internetlivestats.com/one-second/, accessed 15 October 2019.

[169] Mike Isaac and Sheera Frenkel. "Facebook security breach exposes accounts of 50 million users." *New York Times* (Sept. 28, 2018). https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html, accessed 21 October 2019.

[170] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. "Efficient arguments without short PCPs." *IEEE Conference on Computational Complexity*. 2007.

[171] Andras Janosi, William Steinbrunn, Matthias Pfisterer, and Robert Detrano. *Heart Disease Data Set*. https://archive.ics.uci.edu/ml/datasets/Heart+Disease. accessed 15 September 2016. July 22, 1988.

[172] Rob Jansen and Aaron Johnson. "Safely measuring Tor." *ACM Conference on Computer and Communications Security*. 2016.

[173] Marek Jawurek and Florian Kerschbaum. "Fault-tolerant privacy-preserving statistics." *Privacy Enhancing Technologies Symposium*. 2012.

[174] Tobias Jeske. "Floating car data from smartphones: what Google and Waze know about you and how hackers can control traffic." *BlackHat Europe* (2013).

[175] Aaron Johnson. "Design and Analysis of Efficient Anonymous-Communication Protocols." PhD thesis. Yale University, Dec. 2009.

[176] Marc Joye and Benoît Libert. "A scalable scheme for privacy-preserving aggregation of time-series data." *Financial Cryptography*. 2013.

[177] Yael Tauman Kalai and Ran Raz. "Interactive PCP." *International Colloquium on Automata, Languages and Programming*. 2008.

[178] Yael Tauman Kalai and Ron D. Rothblum. "Arguments of proximity." *CRYPTO*. 2015.

[179] Alan F. Karr, Xiaodong Lin, Ashish P. Sanil, and Jerome P. Reiter. "Secure regression on distributed databases." *Journal of Computational and Graphical Statistics* 14.2 (2005).

[180] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-size commitments to polynomials and their applications." *ASIACRYPT*. 2010.

[181] Charlie Kaufman, Paul Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen. *RFC7296: Internet Key Exchange Protocol Version 2 (IKEv2)*. Oct. 2014.

[182] Dogan Kedogan, Dakshi Agrawal, and Stefan Penz. "Limits of anonymity in open environments." *Information Hiding*. 2002.

[183] Josh Keller, K.K. Rebecca Lai, and Nicole Perlroth. "How many times has your personal information been exposed to hackers?" *New York Times* (July 29, 2015).

[184] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments." *ACM Symposium on Theory of Computing*. 1992.

[185] Hartmut Klauck. "Rectangle size bounds and threshold covers in communication complexity." *IEEE Conference on Computational Complexity*. 2003.

[186] Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. "Interactive distributed proofs." *ACM Symposium on Principles of Distributed Computing*. 2018.

[187] Hugo Krawczyk. "Secret sharing made short." *CRYPTO*. 1993.

[188] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. "Privacy-friendly aggregation for the smart-grid." *Privacy Enhancing Technologies Symposium*. 2011.

[189] Eyal Kushilevitz. "Communication complexity." *Advances in Computers*. Vol. 44. Elsevier, 1997.

[190] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is not needed: single database, computationally-private information retrieval." *IEEE Symposium on Foundations of Computer Science*. 1997.

[191] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. "Atom: horizontally scaling strong anonymity." *ACM Symposium on Operating Systems Principles*. 2017.

[192] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. "Riffle." *Proceedings on Privacy Enhancing Technologies*. 2016.

[193] Albert Kwon, David Lu, and Srinivas Devadas. "XRD: scalable messaging system with cryptographic privacy." *USENIX Symposium on Networked Systems Design and Implementation*. 2019.

[194] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Karaoke: distributed private messaging immune to passive traffic analysis." *USENIX Symposium on Operating Systems Design and Implementation*. 2018.

[195] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Yodel: strong metadata security for real-time voice calls." *ACM Symposium on Operating Systems Principles*. 2019.

[196] David Lazar and Nickolai Zeldovich. "Alpenhorn: bootstrapping secure communication without leaking metadata." *USENIX Symposium on Operating Systems Design and Implementation*. 2016.

[197] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. "Towards efficient traffic-analysis resistant anonymity networks." *SIGCOMM*. 2013.

[198] Stevens Le Blond, David Choffnes, Wenxuan Zhou, Peter Druschel, Hitesh Ballani, and Paul Francis. "Towards efficient traffic-analysis resistant anonymity networks." *SIGCOMM*. 2013.

[199] Yehuda Lindell. "Fast cut-and-choose-based protocols for malicious and covert adversaries." *Journal of Cryptology* 29.2 (2016).

[200] Yehuda Lindell and Benny Pinkas. "A proof of security of Yao's protocol for two-party computation." *Journal of Cryptology* 22.2 (2009).

[201] Helger Lipmaa. "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments." *IACR Theory of Cryptography Conference*. 2012.

[202] Barbara Liskov and James Cowling. *Viewstamped replication revisited*. Tech. rep. 2012-021. MIT CSAIL, July 2013.

[203] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic methods for interactive proof systems." *Journal of the ACM* 39.4 (1992).

[204] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. "Fairplay–secure two-party computation system." *USENIX Security Symposium*. 2004.

[205] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. "Sonic: zero-knowledge SNARKs from linear-size universal and updateable structured reference strings." (2019).

[206] Nick Mathewson and Roger Dingledine. "Practical traffic analysis: extending and resisting statistical disclosure." *Privacy Enhancing Technologies Symposium*. 2005.

[207] Luca Melis, George Danezis, and Emiliano De Cristofaro. "Efficient private statistics with succinct sketches." *Network and Distributed System Security Symposium*. Internet Society, Feb. 2016.

[208] Luca Melis, George Danezis, and Emiliano De Cristofaro. "Efficient private statistics with succinct sketches." *Network and Distributed System Security Symposium*. 2016.

[209] Silvio Micali. "CS proofs." *IEEE Symposium on Foundations of Computer Science*. 1994.

[210] Silvio Micali. "Computationally sound proofs." *SIAM Journal of Computing* 30.4 (2000).

[211] Prateek Mittal and Nikita Borisov. "ShadowWalker: peer-to-peer anonymous communication using redundant structured topologies." *ACM Conference on Computer and Communications Security*. 2009.

[212] Payman Mohassel, Mike Rosulek, and Ye Zhang. "Fast and secure three-party computation: the Garbled Circuit approach." *ACM Conference on Computer and Communications Security*. 2015.

[213] Steven J. Murdoch and George Danezis. "Low-cost traffic analysis of Tor." *USENIX Security Symposium*. 2005.

[214] Steven J. Murdoch and Piotr Zieliński. "Sampled traffic analysis by internet-exchange-level adversaries." *Workshop on Privacy Enhancing Technologies*. 2007.

[215] Ellen Nakashima and Barton Gellman. "Court gave NSA broad leeway in surveillance, documents show." *Washington Post* (June 30, 2014).

[216] Moni Naor, Merav Parter, and Eylon Yogev. *The Power of Distributed Verifiers in Interactive Proofs*. 2018.

[217] Moni Naor and Benny Pinkas. "Computationally secure oblivious transfer." *Journal of Cryptology* 18.1 (2005).

[218] Moni Naor, Benny Pinkas, and Omer Reingold. "Distributed pseudo-random functions and KDCs." *EUROCRYPT*. 1999.

[219] National Institute of Standards and Technology. *Specification for the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. Nov. 2001.

[220] C. Andrew Neff. "A verifiable secret shuffle and its application to e-voting." *ACM Conference on Computer and Communications Security*. 2001.

[221] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. "Privacy-preserving matrix factorization." *ACM Conference on Computer and Communications Security*. 2013.

[222] Diego Ongaro and John Ousterhout. "In search of an understandable consensus algorithm." *USENIX Annual Technical Conference*. 2014.

[223] Rafail Ostrovsky and Victor Shoup. "Private information storage." *ACM Symposium on Theory of Computing*. 1997.

[224] Pascal Paillier. "Public-key cryptosystems based on composite degree residuosity classes." *EUROCRYPT*. 1999.

[225] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: nearly practical verifiable computation." *IEEE Symposium on Security and Privacy*. 2013.

[226] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: nearly practical verifiable computation." *Commununications of the ACM* 59.2 (2016).

[227] Robert Douglas Pedersen. *Two-Person Control: a brief history and modern industry practices*. Tech. rep. Livermore, California: Sandia National Laboratory, 2017.

[228] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. "Secure two-party computation is practical." *CRYPTO*. 2009.

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

[230] Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. "VPriv: protecting privacy in location-based vehicular services." *USENIX Security Symposium*. 2009.

[231] Raluca Ada Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. "Privacy and accountability for location-based aggregate statistics." *ACM Conference on Computer and Communications Security*. 2011.

[232] Michael O. Rabin and Ronald L. Rivest. "Efficient end to end verifiable electronic voting employing split value representations." *EVOTE*. 2014.

[233] Charles Rackoff and Daniel R. Simon. "Cryptographic defense against traffic analysis." *ACM Symposium on Theory of Computing*. 1993.

[234] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-round interactive proofs for delegating computation." *ACM Symposium on Theory of Computing*. 2016.

[235] Michael K. Reiter and Aviel D. Rubin. "Crowds: anonymity for web transactions." *ACM Transactions on Information and System Security* 1.1 (1998).

[236] Phillip Rogaway and Mihir Bellare. "Robust computational secret sharing and a unified account of classical secret-sharing goals." *ACM Conference on Computer and Communications Security*. 2007.

[237] Guy N. Rothblum. "Delegating Computation Reliably: Paradigms and Constructions." PhD thesis. Massachusetts Institute of Technology, Sept. 2009.

[238] Guy N. Rothblum and Salil P. Vadhan. "Are PCPs inherent in efficient arguments?" *Computational Complexity* 19.2 (2010).

[239] Len Sassaman, Bram Cohen, and Nick Mathewson. "The Pynchon gate: a secure method of pseudonymous mail retrieval." *Workshop on Privacy in the Electronic Society*. 2005.

[240] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: decentralized anonymous payments from Bitcoin." *IEEE Symposium on Security and Privacy*. 2014.

[241] Claus-Peter Schnorr. "Efficient signature generation by smart cards." *Journal of Cryptology* 4.3 (1991).

[242] Jacob T. Schwartz. "Fast probabilistic algorithms for verification of polynomial identities." *Journal of the ACM* 27.4 (1980).

[243] Andrei Serjantov, Roger Dingledine, and Paul Syverson. "From a trickle to a flood: active attacks on several mix types." *Information Hiding*. 2002.

[244] Srinath Setty. "Spartan: efficient and general-purpose zkSNARKs without trusted setup" (2019).

[245] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. "Resolving the conflict between generality and plausibility in verified computation." *EuroSys*. ACM. 2013.

[246] Srinath Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. "Making argument systems for outsourced computation practical (sometimes)." *Network and Distributed System Security Symposium*. 2012.

[247] Adi Shamir. "How to share a secret." *Communications of the ACM* 22.11 (1979).

[248] Adi Shamir. "IP = PSPACE." *Journal of the ACM* 39.4 (1992).

[249] Elaine Shi, T.H. Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. "Privacy-preserving aggregation of time-series data." *Network and Distributed System Security Symposium*. 2011.

[250] Victor Shoup. "A proposal for an ISO standard for public key encryption (version 2.1)" (2001).

[251] Victor Shoup. "OAEP reconsidered." *CRYPTO*. 2001.

[252] Emin Gün Sirer, Sharad Goel, Mark Robson, and Doğan Engin. "Eluding carnivores: file sharing with strong anonymity." *ACM SIGOPS European Workshop*. ACM. 2004.

[253] Ben Smith. *Uber Executive Suggests Digging Up Dirt On Journalists*. https://www.buzzfeed.com/bensmith/uber-executive-suggests-digging-up-dirt-on-journalists. accessed 20 September 2016. Nov. 17, 2014.

[254] Madhu Sudan. "Probabilistically checkable proofs." *Communications of the ACM* 52.3 (2009).

[255] Paul Syverson. "Why I'm not an entropist." *Security Protocols XVII*. 2013.

[256] Justin Thaler. "Time-optimal interactive proofs for circuit evaluation." *CRYPTO*. 2013.

[257] Ken Thompson. "Reflections on trusting trust." *Communications of the ACM* 27.8 (1984).

[258] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. "Adnostic: privacy preserving targeted advertising." *Network and Distributed System Security Symposium*. 2010.

[259] Liam Tung. "Facebook is using your 2FA phone number to target ads at you" (Oct. 1, 2018). https://www.zdnet.com/article/facebook-is-using-your-2fa-phone-number-to-target-ads-at-you/, accessed 21 October 2019.

[260] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. "Stadium: a distributed metadata-private messaging system." *ACM Symposium on Operating Systems Principles*. 2017.

[261] U.S. Department of Health and Human Services. *AIDSinfo*. https://aidsinfo.nih.gov/apps. 2019.

[262] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. "Vuvuzela: scalable private messaging resistant to traffic analysis." *ACM Symposium on Operating Systems Principles*. 2015.

[263] Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. "An analysis of social network-based Sybil defenses." *SIGCOMM* (2010).

[264] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, abhi shelat, Justin Thaler, Michael Walfish, and Thomas Wies. "Full accounting for verifiable outsourcing." *ACM Conference on Computer and Communications Security*. 2017.

[265] Riad S. Wahby, Srinath T.V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. "Efficient RAM and control flow in verifiable outsourced computation." *Network and Distributed System Security Symposium*. 2016.

[266] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. "Doubly-efficient zkSNARKs without trusted setup." *IEEE Symposium on Security and Privacy*. 2018.

[267] Michael Waidner and Birgit Pfitzmann. "The Dining Cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure serviceability." *EUROCRYPT*. 1989.

[268] Michael Walfish and Andrew J. Blumberg. "Verifying computations without reexecuting them." *Communications of the ACM* 58.2 (2015).

[269] Declan Walsh. "Dilemma for Uber and rival: Egypt's demand for data on riders." *New York Times* (June 10, 2017). https://www.nytimes.com/2017/06/10/world/middleeast/egypt-uber-sisi-surveillance-repression-careem.html, accessed 21 October 2019.

[270] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y. Zhao. "Defending against Sybil devices in crowdsourced mapping services." *ACM International Conference on Mobile Systems, Applications, and Services*. ACM. 2016.

[271] Stanley L. Warner. "Randomized response: a survey technique for eliminating evasive answer bias." *Journal of the American Statistical Association* 60.309 (1965).

[272] Richard Ryan Williams. "Strong ETH breaks with Merlin and Arthur: short non-interactive proofs of batch evaluation." *IEEE Conference on Computational Complexity*. 2016.

[273] William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian. *Wisconsin Prognostic Breast Cancer Data Set*. https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/. accessed 15 September 2016. Dec. 1995.

[274] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. "Dissent in numbers: making strong anonymity scale." *USENIX Symposium on Operating Systems Design and Implementation*. 2012.

[275] David Isaac Wolinsky, Ewa Syta, and Bryan Ford. "Hang with your buddies to resist intersection attacks." *ACM Conference on Computer and Communications Security*. 2013.

[276] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. "Libra: succinct zero-knowledge proofs with optimal prover computation." *CRYPTO*. 2019.

[277] Andrew C. Yao. "Protocols for secure computations." *IEEE Symposium on Foundations of Computer Science*. 1982.

[278] Andrew Chi-Chih Yao. "How to generate and exchange secrets." *IEEE Symposium on Foundations of Computer Science*. 1986.

[279] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. "SybilLimit: a near-optimal social network defense against Sybil attacks." *IEEE Symposium on Security and Privacy*. 2008.

[280] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. "SybilGuard: defending against Sybil attacks via social networks." *SIGCOMM*. ACM. 2006.

[281] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. "A zero-knowledge version of vSQL" (2017).

[282] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. "vSQL: verifying arbitrary SQL queries over dynamic outsourced databases." *IEEE Symposium on Security and Privacy*. 2017.

[283] Richard Zippel. "Probabilistic algorithms for sparse polynomials." *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*. 1979.