LIGHTWEIGHT PRIVACY FOR AND FROM THE MASSES

Yiping Ma

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2025

Supervisor of Dissertation	Co-Supervisor of Dissertation
Sebastian Angel	Tal Rabin
Raj and Neera Singh Assistant Professor, Computer and Information Science	Rachleff Family Professor, Computer and Information Science

Graduate Group Chairperson

Mayur Naik, Misra Family Professor, Computer and Information Science

Dissertation Committee

Brett Hemenway Falk, Research Assistant Professor, Computer and Information Science
Yuval Ishai, Professor, Computer Science, Technion – Israel Institute of Technology
Sampath Kannan, Henry Salvatori Professor, Computer and Information Science
Pratyush Mishra, Assistant Professor, Computer and Information Science
Antigoni Polychroniadou, Executive Director, JP Morgan AI Research

Dedicated to my loved ones.

ACKNOWLEDGEMENT

I often joke with my friends that magical relationships begin with randomness. It struck me today that this isn't a joke. Yet beyond the magic of how things start lies something even more powerful—the quiet, lasting impact on me from relationships with many wonderful people. I am truly grateful to all of you.

In 2018, I was coincidentally introduced to the field of cryptography by Xiao Wang and Ruiyu Zhu. Xiao later introduced me to a winter school in Shenzhen and the lectures there absolutely fascinated me. I want to thank them for their friendliness and the opportunity they opened up for me, which ultimately led me to pursue a PhD.

My two advisors made my PhD journey truly the best it could be. When I was a junior student who knew little about research and academic writing, they gave me tons of feedback on my papers and talks, taught me skills in all aspects with great patience, and were incredibly forgiving when I made mistakes. More important than all of that, they were there to support me during the most difficult time of my PhD. Each of you has guided me in your own way and shaped who I am.

Sebastian, you pushed me to be a clearer and braver thinker. You always made sure that I was asking myself the right questions at each junction I faced, from choosing research problems to making career decisions. I admire your dedication to the quality of research and your perseverance to work towards challenging goals, and I hope to emulate them in the future.

Tal, I owe you more than I can say. You are an oracle that hands me the most helpful answers when I come to you with my hardest questions. I've learned so much from working with you, from how you think about a problem to how you envision the bigger picture of research; and beyond that, you are a friend, a life mentor, and a true role model whom I hope I can live up to one day. Most importantly, the fact that you have confidence in me has made me believe that it is possible to achieve my goals that I once thought were too far away.

Collaboration is one of the best parts in my PhD. I was extremely lucky to start collaboration

with Yuval Ishai in my second year. Yuval has a gift for discovering unexpected connections across different ideas, and working with him changed the way I think about research. I truly appreciated how incredibly fast and thorough his feedback always was. Beyond that, I continue to admire his great sense of humor, humility, and optimism.

My two visits to Technion turned out to be wonderful coincidences—they brought me together with people who later became collaborators and friends. I want to thank all of you for making the visits fruitful and colorful: Amit Agarwal, Elette Boyle, Niv Gilboa, Matan Hamilis, Xin Huang, Mahimna Kelkar, Victor Kolobov, Daniel Lee, Varun Narayanan, and Yaxin Tu. The second part of this thesis stems from the work I did during the first visit with Mahimna, the time in Haifa together with you is one of the best memories during my PhD. With the privacy team at Google, we made a great follow-up work to the second part of this thesis. I want to thank Adrià Gascón, Baiyu Li, and Mariana Raykova for their efforts in making this possible.

I had two wonderful summers with the cryptography group at JP Morgan AI Research and the fellow interns: Alex Bienstock, Daniel Escudero, Yue Guo, Harish Karthikeyan, Lisa Masserova, Nikolas Melissaris, Antigoni Polychroniadou, Akira Takahashi, and Chenkai Weng. The first part of this thesis comes out of a collaboration with Antigoni. She has always been encouraging, forgiving, and a strong advocate for me, for which I am genuinely grateful. I was also fortunate to work with people in AWS cryptography group: Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk; they are welcoming, kind, and so much fun to work with.

Conference and talk travels have been an exciting part of PhD life. I want to thank Jessica Chen, Alexandra Henzinger, Darya Kaviani, Ryan Lehmkuhl, Lisa Masserova, Peihan Miao, Mayank Rethee, Wenhao Zhang and Jinhao Zhu for hosting me for talks. I thank Yingchen Wang for sharing rooms with me at many conferences—somehow, we always ended up attending the same ones!

Being with fellow students at Penn has given me a true sense of belonging. DSL, Levine 613, and AGH 342 would not be such fun places without you: Lef Ioannidis, Eli Margolin, Matan Shtepel,

Tushar Mopuri, Alireza Shirzad, Jess Woods, Haoran Zhang, Yuxuan Zhang, Ke Zhong. Jess and Ke, you have always been kind and reliable research buddies. Eli, Lef, and Haoran—you light up the office space with your great sense of humor and insightful conversations. Yuxuan, you are such a quiet and sweet person to share an office with. Tushar and Ali created a collaborative, interactive vibe in the lab; and I especially thank Ali for continuing to run the security seminar when I was tied up with other responsibilities. Matan, your vibrant energy and enthusiasm for research are truly contagious.

The faculty I met at Penn have been extremely supportive, and the friendly environment in the CIS department has made me feel at home. I am especially thankful to Brett Hemenway, Sampath Kannan, Pratyush Mishra for serving as my committee members.

I also want to thank the staff at Penn: Britton Carnevali, Cheryl Hickey, Ally Moraschi, and Mari Thach. Room bookings, catering for seminars, office moving and on and on; they took care of it all with such professionalism.

PhD life would have been so much harder without friends in Philly and my hometown. I want to thank everyone of you for supporting me: Yifan Cai, Danfeng Cao, Peng Cao, Xinyi Chen, Danning He, Anran Huang, Van Truong, Wenshi Wang, Yifan Wu, and Ke Zhong. Xinyi and Anran have been sharing apartment with me for three years. Xinyi is a creative source of ideas for cooking, traveling, and even giving talks. I am grateful for all the fun days and conversations we shared. Anran generously provided me support when I first moved to Philly. I am also thankful for her continuous encouragement over the past years in pursuing my career goals.

Finally, I want to thank my family for their support. I appreciate the freedom my parents have given me to pursue anything I want, and I am grateful for my grandma who played a significant role in shaping the person I am today, instilling in me the values of kindness, commitment, and hard work.

ABSTRACT

LIGHTWEIGHT PRIVACY FOR AND FROM THE MASSES

Yiping Ma

Sebastian Angel

Tal Rabin

Online services today rely on a massive amount of user data. Yet, the data that users supply to or fetch from the services expose their personal information, which often in practice leads to privacy failures. In this dissertation, we design protocols and build systems that allow users to supply or fetch data without putting their privacy at risk. While this is achievable in theory with general-purpose cryptography tools, applying them at the scale of today's applications—often serving millions of users—is prohibitively expensive. Our insight is that the large user base can be leveraged to get lightweight privacy, although it is often seen as a performance bottleneck.

We consider two types of problems under the model of a central "powerful" server and many "weak" clients:

- How does the server aggregate (or more advanced, train machine learning models on) private data of clients without learning any individual client's data? Here, the clients "push" private data to the server.
- How do the clients fetch data from a public database at the server while completely hiding from the server which data they want to fetch? Here, the clients privately "pull" data from the server.

For the first problem, we designed and built two systems: 1) Flamingo, a secure aggregation system for high-dimensional vector inputs that can be used to train neural networks on private data across hundreds of thousands of clients; 2) Armadillo, a system that shares Flamingo's properties but additionally offers disruption resistance against adversarial clients. The key design principles underlying

both systems are distributing the trust among the large number of clients and leveraging them to assist with secure computation. The main challenge we addressed was making clients lightweight enough to run on weak devices.

The second problem is closely related to a classical cryptography problem called Private Information Retrieval (PIR). We consider PIR under "the shuffle model", where queries can be made anonymously by many clients. Under this model, we give the first single-server PIR with information-theoretic security and sublinear communication per query.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

Modern applications serve millions of users, collecting and processing vast amounts of personal data to power services such as web search, social media, fraud detection, and more. Ensuring the privacy of such data is a fundamental challenge, especially at this scale.

The traditional approach to designing privacy-protecting systems relies heavily on general-purpose cryptographic tools developed by the cryptography community. Over the past decade, significant effort has gone into optimizing these generic tools—such as secure multi-party computation (MPC) and private information retrieval (PIR) that are concerned in this work—in increasingly powerful ways. These tools often come with strong formal guarantees and have demonstrated remarkable performance at small scale.

However, when applying them in the real world, we encounter a fundamental mismatch between the scale of today's applications and what these tools were originally expected to handle—even the most optimized versions become impractical at such scale. For instance, MPC was mostly developed and benchmarked under the implicit expectation of running on up to tens of parties. Even quadratic communication would become infeasible for a few thousand parties. Similarly, while PIR has been optimized to the point where a query to large databases can be answered at reasonable cost, real-world systems often receive thousands to millions of requests per second. All in all, the large user base appears to be the barrier for deploying privacy solutions at scale.

In this dissertation, we explore a different approach: can we leverage the availability of many users to design privacy-protecting systems? We answer this question with lightweight and scalable solutions that meet practical demands; a salient example demonstrated in this dissertation is the feasibility of training neural networks on private image data owned by tens of thousands of clients.

This dissertation is in two parts. Both parts share the same setting: a central server and many clients. The server abstracts the untrusted platforms that provide online services, and the clients

abstract users' mobile devices, such as laptops or cell phones. The server and the clients interact to securely perform two types of tasks: in the first, the clients privately "push" their data to the server; and in the second part, the clients privately "pull" data from the server. We describe these tasks in more detail next.

Part I: Private "push". Online service providers often explicitly ask users for their data, and in exchange, they use the data to improve user experience. In such scenarios, users may still wish to share their data in order to enjoy better services, but seek to do so in a manner that minimizes privacy risks.

In Part I, we focus on a problem called *secure aggregation*: how can a server compute the sum of inputs from a set of clients without learning any individual input? While summation may appear too simple for complex applications, it suffices for computing a wide range of statistics [CGB17, HIKR23], including max/min, variance, histograms, and as this work shows, even for training machine learning models.

The secure aggregation problem has received considerable study in recent years, but the setting we consider in this work—federated learning—introduces new and significant challenges. In this setting, a server needs to consecutively run aggregations many times with a large set of computationally weak, unstable, and potentially faulty clients. This dissertation builds systems that provide fault tolerance while remaining efficient enough to support many rounds of secure aggregation on many clients. The observation we made is that most users in federated learning scenarios are honest and mutually unassociated, and at any given time a majority of them are available. We exploit this observation by splitting trust among the clients and leveraging whoever is present to assist with secure computation. However crucially, this must be done in a lightweight way respecting the limited capabilities of client devices, and a robust way respecting to the fact that they are unstable and error-prone.

At a high level, our systems work as follows: First, each client sends their input in some masked form to the server; then, a small committee—sampled randomly from the client population—is

responsible for unmasking the aggregation result. In short, regular clients contribute their inputs and then exit; the committee takes on slightly more computation and communication to complete the aggregation, in which their cost is only proportional to the committee size (much smaller than the size of the whole population). As long as a majority of the committee clients stay online at the time of unmasking, the aggregation can be completed.

Part II: Private "pull". In this part, we focus on a cryptographic primitive called *private information retrieval* (PIR). In PIR, a server holds a database of n bits and a client can issue a query to retrieve the i-th bit from the database while hiding i from the server. This simple primitive recently becomes useful in security and systems communities: researchers have built many privacy-preserving systems using PIR in a blackbox way, such as metadata-hiding communication [AS16, AYA+21], encrypted search systems [DFL+20], private search engines [HDCGZ23], and more. The efficiency of these systems relies on the efficiency of the underlying PIR schemes, and recent years have seen significant research efforts toward developing faster PIR protocols.

This dissertation considers PIR problem under *the shuffle model*: we assume there are many clients making anonymous queries to the database simultaneously. Essentially, in addition to the database server, we assume there is a shuffler who permutes the queries before they reach the database and permutes back the responses for the clients. With shuffling, privacy for clients can be achieved in a fundamentally different way from the classical PIR schemes [IKOS06]. This leads to a property in our PIR construction that, as the number of simultaneous queries grows, the server's per-query cost actually decreases—an advantage not present in the classical PIR model.

We demonstrate the power of the shuffle model by achieving two security and efficiency balances for PIR that are provably impossible in the classical model:

- There exists PIR with sublinear communication and information-theoretic security under the shuffle model.
- In the shuffle model, we do not need any padding for PIR database records, as opposed to the classical model where every record is padded to the maximum record length.

Our follow-up work [GIK+24] further validates this direction with a construction in which the server can answer hundreds of queries per second to a database of several gigabytes.

1.1 Overview of results

This section provides a brief background and an overview of our main results.

1.1.1 Secure aggregation in federated learning

The earlier introduction summarized the shared design principles of the two systems in this dissertation: Flamingo and Armadillo. In this part, we focus on the similarities and differences in their guarantees and the resulting performance trade-offs.

We aim for the following security and efficiency properties when we design secure aggregation systems for the federated learning setting.

Privacy. We aim for privacy against the server, in the sense that it learns only the sum of client's inputs but nothing else. This implies that the server learns nothing about each individual client's input. We say a protocol provides privacy against an honest-but-curious server if the above property holds when the server follows the protocol but attempts to infer additional information from the messages it sees. The protocol achieves privacy against a malicious server if the same guarantee holds even when the server behaves arbitrarily.

Fault tolerance. We consider two types of faults: passive and active. Passive faults refer to clients that follow the protocol but may drop out of the execution at any time (e.g., due to power loss or network disconnection). Active faults involve arbitrary misbehavior, such as submitting malformed inputs, omitting messages, or sending messages that deviate from the protocol. We say a protocol that tolerates (passive or active) faults if the server always outputs a correct sum result, regardless of which coalition of clients are faulty, as long as the number of faulty clients remains below a certain threshold.

Multi-iteration support. Federated learning requires a server to interact with clients hundreds of iterations to train a model; each iteration involves one execution of secure aggregation on hundreds

to thousands of clients [KMA+21]. Prior to our work, even the best secure aggregation protocols [BBG+20], though tailored to meet constraints of the weak clients, remained infeasible for a complete training process. The key limitation is that these protocols were designed to handle only a single execution of aggregation; specifically, they follow a pattern of having each client set up input-independent secrets with several other clients, and then computing a single sum using the secrets. These secrets cannot be reused for privacy reasons. Consequently, for each aggregation in the training process, all clients need to perform an expensive fresh setup. As a result, using these protocols for a complete training is infeasible. We aim for protocols with the ability to perform setup just once, after which the clients and the server can run many aggregations continuously.

Lightweight client. Suppose each client has an input of size ℓ, and there are n clients participating in an aggregation for a sum. Our goal is to keep the client-side computation within $O(n + \ell)$. This rules out the trivial MPC solution—having each client secret-share its input with all others—which incurs $O(n\ell)$ computation per client. As a reference point, n is typically on the order of a few hundred to a few thousand for each aggregation, and ℓ can be as large as 500K [KMA+21].

Both of our systems have multi-iteration support and lightweight client computation; for each aggregation most of the clients do $O(\mathsf{polylog}(n) + \ell)$ work and a small number of clients take $O(n)$ work. The distinction lies in fault tolerance: Flamingo tolerates passive dropouts, whereas Armadillo resists active faults. Specifically, Armadillo supports validation of L_2 and L_∞ norms of client inputs, and guarantees output delivery. As a trade-off, it incurs higher computational overhead on the clients and the server compared to Flamingo. In terms of privacy, Flamingo provides privacy against a malicious server with abort while Armadillo protects against an honest-but-curious one.

1.1.2 PIR in the shuffle model

PIR allows a client to get a bit from a database $x = \{0, 1\}^n$ while hiding from the database which bit is being accessed. A trivial solution for PIR is to let the client download all the n bits regardless which bit they want to get. This has perfect security but requires communication linear in the database size. Unfortunately, perfect security with sublinear communication is provably impossible with only one database server [CGKS95]. Consequently, meaningful information-theoretic PIR

constructions all work under the multi-server model, where databases are replicated among multiple non-colluding servers; the client communicates with each server sublinear number of bits to get the desired bit.

The multi-server model, when deployed in practice, has several drawbacks. These include storage overhead for database replicas, the need to keep databases synchronized across servers, and the difficulty to justify the non-collusion assumption in practice: often, the PIR service provider owns and operates the database and its replicas, and hence they can see the queries made to all the servers.

Many recent single-server PIR protocols have been proposed [MW22, ACLS18], but they still fall short of the performance achieved by multi-server PIR. While a few schemes come close to the efficiency of multi-server PIR, they incur substantial client-side storage overhead [HHCG$^+$23, ZPSZ23].

PIR in the shuffle model introduces a new design space. We assume that queries are permuted by a primitive that implements a two-way anonymous communication channel between the database and the clients. Then we compile a multi-server PIR protocol into PIR with a single database in the shuffle model: each client randomizes its query index into a small number of sub-queries by locally running a multi-server PIR query algorithm. Then, all sub-queries are anonymously sent to the server, who responds to each sub-query separately without knowing which client it originates from. Privacy against the database is achieved by shuffling the sub-queries, and in this work particularly, we show that this privacy guarantee can be information-theoretic.

This shuffle model conceptually is almost equivalent to the two-server trust model (i.e., we have a database server and a shuffle server and they do not collude), but with a key distinction: the shuffle server does not hold a copy of the database, thereby eliminating the need for database replication that is required in the classical two-server setting. Now, a salient benefit in deployment is that the shuffle server can be operated by an entity independent of the database owner; and can even be operated in a decentralized manner by many volunteers worldwide like Tor.

Below we summarize our main results. The first result is a class of PIR protocols that have sublinear

communication and information-theoretic security in the shuffle model, given in an informal theorem below. Our construction does not require any state information on the client side. Following one-time preprocessing at the server, we also get sublinear server computation per query.

Theorem 1.1.1 (Informal)**.** *For every constant* $0 < \gamma < 1$ *, there is a single-server PIR protocol in the shuffle model that, on database of size* n*, has* $O(n^\gamma)$ *per-query computation and communication with* $1/\mathsf{poly}(n)$ *statistical security, when* $\mathsf{poly}(n)$ *clients simultaneously access the database.*

Our second result is about an orthogonal use case of shuffle model, this enables PIR with variable-sized records. Real world databases typically contain records in a variety of sizes and it is often important to hide not just the record but also its size. Unfortunately, in the classical PIR setting, nothing better can be done apart from padding each record so that they are all now of the same length; this poses an undue communication cost on the majority of clients who may only wish to retrieve small records. By considering PIR in the shuffle model, we show how the size of any individual retrieved record can be hidden without any padding even where there are only a small number of querying clients. In particular, we show a novel recursive approach for splitting a record of size ℓ to be retrieved which incurs only $\Theta(\mathsf{polylog}(\ell))$ overhead over the insecure solution.

1.2 Roadmap

Chapter 2 focuses on secure aggregation in federated learning. It provides background for federated learning and presents two aggregation systems Flamingo (§2.2) and Armadillo (§2.3). For each system, we present protocol design, security properties, implementation and evaluation. We also discuss limitations of both systems. Chapter 3 turns to PIR in the shuffle model. We give a formal definition of PIR problem under this model (§3.3), and present a generic compiler (§3.5) that transforms classical multi-database PIR protocols to single-database PIR in the shuffle model, and provide concrete instantiations (§3.5.4) that achieve the complexity stated in Theorem 3.1.1. Then, we present a construction for handling variable-sized records in PIR (§3.6). Finally, we discuss open problems and related work (§3.7).

CHAPTER 2

"PRIVATE PUSH": SECURE AGGREGATION

2.1 Introduction

Applications today rely on usage data from a massive number of clients. A single fraud detection platform monitors spending of millions of customers worldwide to build their detection model [Cao24, FIC24]. A single social app collects from billions of users their demographic information, activity data, and more for customized advertising [Fed24]. We abstract these examples as a single server aggregating data from many clients, where the aggregation can be as simple as computing a mean statistic, or as complex as training a machine learning model.

In this context we ask two questions: Can each client keep their own data hidden even if a single untrusted server coordinates the entire aggregation? Can the server always get a valid aggregation result even when arbitrary faults happen at any client? We further put these questions in a challenging setting motivated by recent emergence of federated learning: *a central powerful server (e.g., a computing cluster) interacts with many weak clients (e.g., millions of mobile devices)*; here "powerful vs. weak" is with respect to computational power, communication capacity, and device availability. In this work, we answer these questions affirmatively with two secure aggregation systems, Flamingo and Armadillo. Flamingo deals with passive faults and Armadillo deals with active disruptions. Furthermore, we demonstrate that Flamingo, when combined with federated learning mechanism, can train a model of 500K parameters on private data across tens of thousands of clients, without the server learning any individual client's training data.

2.2 Flamingo: Efficient multi-run secure aggregation with one-time setup

2.2.1 Motivation

In federated learning, the server and the clients interact iteratively to train a model. The server in each training iteration randomly selects a subset of clients from the whole training population, and sends them the current model's weights (the model starts with random weights). Each selected

client updates the model's weights by running a prescribed training algorithm on its data *locally*, and then sends the updated weights to the server. The server updates the model by averaging the collected weights. The training takes multiple such iterations until the model converges.

This distributed training pattern is introduced with the goal of providing a critical privacy guarantee in training—the raw data never leaves the clients' devices. However, prior works [MSCS19, ZLH19] show that a client's individual model still leak information about its raw data, which highlights the need for a mechanism that can securely aggregate the weights from clients [MMR+17, YM20]. This is precisely an instance of secure aggregation.

Many protocols and systems for secure aggregation have been proposed, e.g., in the scenarios of private error reporting and statistics collection [SCR+11, CSS11, CGB17, ACD+21, BBCG+21, ZMA22]. However, secure aggregation in federated learning, due to its specific model, faces unprecedented challenges: large-sized inputs (e.g., model weights), multiple rounds of aggregation prior to model convergence, a large number of participants who are unstable (i.e., some devices might go offline prior to completing the protocol) and some of them may even be disruptive (e.g., a malicious device sends bogus messages to disrupt the aggregation result). It is therefore difficult to directly apply these protocols in a black-box way and still get good guarantees and performance.

Recent works [BIK+17, BBG+20, SHY+22] propose secure aggregation tailored to federated learning scenarios. In particular, a state-of-the-art protocol [BBG+20] (which we call BBGLR) can handle one aggregation with thousands of clients and high-dimensional input vectors, while tolerating devices dropping out at any point during their execution. The drawback of these protocols is that they only examine aggregation in one training iteration in the full training process, i.e., a selection of a subset of the clients and a sum over their inputs.

Utilizing the BBGLR protocol (or its variants) multiple times to do summations in the full training of a model incurs high costs. Specifically, these protocols follow the pattern of having each client establish input-independent secrets with several other clients in a *setup* phase, and then computing a single sum in a *collection* phase using the secrets. These secrets cannot be reused for privacy reasons.

Consequently, for each execution of aggregation in the training process, one needs to perform an expensive fresh setup. Furthermore, in each *round* (client-server round trip) of the setup and the collection phases, the server has to interact with all of the clients. In the setting of federated learning, such interactions are especially costly as clients may be geographically distributed and may have limited compute and battery power or varying network conditions.

2.2.2 Summary of contributions

In this paper we propose *Flamingo*, the first single-server secure aggregation system that works well for multiple executions of aggregation and that can support full sessions of training in the stringent setting of federated learning. At a high level, Flamingo introduces a one-time setup and a collection procedure for computing multiple sums, such that the secrets established in the setup can be reused throughout the collection procedure. For each summation in the collection procedure, clients mask their input values (e.g., updated weight vectors in federated learning) with a random mask derived from those secrets, and then send the masked inputs to the server. The server sums up all the masked inputs and obtains the final aggregate value (what the server wants) masked with the sum of all masks. Flamingo then uses a lightweight protocol whereby a small number of randomly chosen clients (which we call *decryptors*) interact with the server to remove the masks.

The design of Flamingo significantly reduces the overall training time of a federated learning model compared to running BBGLR multiple times. First, Flamingo eliminates the need for a setup phase for each summation, which reduces the number of rounds in the full training session. Second, for each summation, Flamingo only has one round that requires the server to contact all of the clients (asking for their inputs); the rest of the interactions are performed between the server and a few clients who serve as decryptors.

Besides training efficiency, the fact that a client needs to only speak once in an aggregation reduces the model's bias towards results that contain only data from powerful, stably connected devices. In Flamingo, the server contacts the clients once to collect inputs; in contrast, prior works have multiple client-server communication rounds in the setup phase (before input collection) and thus filter out weak devices for summation, as staying available longer is challenging. Seen from a different angle,

if we fix the number of clients, the failure probability of a client in a given round, and the network conditions, Flamingo's results are of higher quality, as they are computed over more inputs than prior works.

In summary, Flamingo's technical innovations are:

- *Lightweight dropout resilience.* A new mechanism to achieve dropout resilience in which the server only contacts a small number of clients to remove the masks. All other clients are free to leave after the one round in which they submit their inputs without harming the results.
- *Reusable secrets.* A new way to generate masks that allows the reuse of secrets across multiple times of aggregation.
- *Per-iteration graphs.* A new graph generation procedure that allows clients in Flamingo to unilaterally determine (without the help of the server as in prior work) which pairwise masks they should use in any given iteration.

These advancements translate into significant performance improvements (§2.2.10). For a 10-iteration pure summation task, Flamingo is 3× faster than BBGLR (this includes Flamingo's one-time setup cost), and includes the contribution of more clients in its result; this performance is measured under the assumption that BBGLR can set up a trusted key directory (e.g., PKI) for free for each aggregation. Therefore, our improvement in practice over BBGLR should be far more pronounced. When training a neural network on the Extended MNIST dataset, Flamingo takes about 40 minutes to converge while BBGLR needs roughly 3.5 hours to reach the same training accuracy.

2.2.3 Setting, threat model and goals

Secure aggregation is useful in a variety of domains: collecting, in a privacy-preserving way, error reports [CGB17, BBCG$^+$21], usage statistics [SCR$^+$11, CSS11], and ad conversions [ZMA22, ACD$^+$21]; it has even been shown to be a key building block for computing private auctions [ZMMA23]. But one key application is *secure federated learning*, whereby a server wishes to train a model on

data that belongs to many clients, but the clients do not wish to share their data (or other intermediate information such as weights that might leak their data [ZLH19]). To accomplish this, each client receives the original model from the server and computes new *private* weights based on their own data. The server and the clients then engage in a secure aggregation protocol that helps the server obtain the sum of the clients' private weights, without learning anything about individual weights beyond what is implied by their sum. The server then normalizes the sum to obtain the average weights which represent the new state of the model. The server repeats this process until the training converges.

This process is formalized as follows. Let $[z]$ denote the set of integers $\{1, 2, \ldots, z\}$, and let $\mathbf{x}$ denote a vector; all operations on vectors are component-wise. A total number of N clients are fixed before the training starts. Each client is indexed by a number in $[N]$. The training process consists of T iterations. In each iteration $t \in [T]$, a set of clients is randomly sampled from the N clients, denoted as S_t. Each client $i \in S_t$ has an input vector, $\mathbf{x}_{i,t}$, for iteration t. (A client may be selected in multiple iterations and may have different inputs for those iterations.) In each iteration, the server wants to securely compute the sum of the $|S_t|$ input vectors, $\sum_{i \in S_t} \mathbf{x}_{i,t}$.

In practical deployments of federated learning, a complete sum is hard to guarantee, as some clients may drop out in the middle of the aggregation process and the server must continue the protocol without waiting for them to come back (otherwise the server might be blocked for an unacceptable amount of time). So the real goal is to compute the sum of the input vectors from the largest possible subset of S_t; we elaborate on this in the next few sections.

Target deployment scenario

Based on a recent survey of federated learning deployments [KMA+21], common parameters are as follows. N is in the range of 100K–10M clients, where $|S_t| = 50$–5,000 clients are chosen to participate in a given round t. The total number of rounds T for a full training session is 500–10,000. Input weights ($\mathbf{x}_{i,t}$) have typically on the order of 1K–500K entries for the datasets we surveyed [KNH, Kri09, CDW+18, CDW+].

Clients in these systems are heterogeneous devices with varying degrees of reliability (e.g., cellphones, servers) and can stop responding due to device or network failure.

Communication model

Each client communicates with the server through a private and authenticated channel. Messages sent from clients to other clients are forwarded via the server, and are end-to-end encrypted and authenticated.

Failure and threat model

We model failures of two kinds: (1) honest clients that disconnect or are too slow to respond as a result of unstable network conditions, power loss, etc; and (2) arbitrary actions by an adversary that controls the server and a bounded fraction of the clients. We describe each of these below.

Dropouts. For each summation, the server interacts with the clients (or a subset of them) several times (several *rounds*). If a server initiates a round and contacts a set of clients, but some of the clients are unable to respond in a timely manner, the server has no choice but to keep going; the stragglers are dropped and do *not* participate in the rest of the computation for this summation. In practice, the fraction of dropouts in the contacted set depends on the client response time distribution and a server timeout (e.g., one second); longer timeouts mean lower fraction of dropouts.

There are two types of clients in the system: regular clients that provide their input, and *decryptors*, who are special clients whose job is to help the server recover the final result. We upper bound the fraction of regular clients that drop out in any given aggregation round by δ, and upper bound the fraction of decryptors that drop out in any given aggregation round by δ_D.

Adversary. We assume a static, malicious adversary that corrupts the server and up to an η fraction of the total N clients. That is, the adversary compromises $N\eta$ clients independent of the protocol execution and the corrupted set stays the same throughout the entire execution (i.e., all training iterations). Note that malicious clients can obviously choose to drop out during protocol execution, but to make our security definition and analysis clear, we consider the dropout of malicious clients separate from, and in addition to, the dropout of honest clients.

Similarly to our dropout model, we distinguish between the fraction of corrupted regular clients (η_{S_t}) and corrupted decryptors (η_D). Both depend on η but also on how Flamingo samples regular clients and decryptors from the set of all N clients. We defer the details to Appendix A.1, but briefly, $\eta_{S_t} \approx \eta$; and given a (statistical) security parameter κ, η_D is upper bounded by $\kappa\eta$ with probability $2^{-\Theta(\kappa)}$.

Threshold requirement. The minimum requirement for Flamingo to work is $\delta_D + \eta_D < 1/3$. For a target security parameter κ, we show in Appendix A.3 how to select other parameters for Flamingo to satisfy the above requirement and result in minimal asymptotic costs.

Remark 1 (Comparison to prior works on threat model). BBGLR and other works [BIK+17, BBG+20] also have a static, malicious adversary but only for a single execution of aggregation. In fact, in Section 2.2.5 we show that their protocol cannot be naturally extended to multiple aggregations that withstands a malicious adversary throughout.

Properties

Flamingo is a secure aggregation system that achieves the following properties under the above threat and failure model. We give informal definitions here, and defer the formal definitions to Section 2.2.8.

- *Dropout resilience*: When all parties follow the protocol, the server, in each iteration t, will get a sum of inputs from all the online clients in S_t. Note that this implicitly assumes that the protocol both completes all the iterations and outputs meaningful results.
- *Security*: For each iteration t summing over the inputs of clients in S_t, a malicious adversary learns the sum of inputs from at least $(1 - \delta - \eta)|S_t|$ clients.

Besides the above, we introduce a new notion that quantifies the quality of a single sum.

- *Sum accuracy*: An execution of summation has sum accuracy τ if the final sum result contains the contribution of a τ fraction of the clients who are selected to contribute to that summation (i.e., $\tau|S_t|$).

Remark 2 (Input correctness). In the context of federated learning, if malicious clients input bogus weights, then the server could derive a bad model (it may even contain "backdoors" that cause the model to misclassify certain inputs [BVH+20]). Ensuring correctness against this type of attack is out of the scope of this work; to our knowledge, providing strong guarantees against malicious inputs remains an open problem. Some works [RNFH19, RZHP20, RNM+21, CGJvdM22, ABIW22, BGL+22] use zero-knowledge proofs to bound how much a client can bias the final result, but they are unable to formally prove the absence of all possible attacks.

Remark 3 (Comparison to prior work on privacy guarantee). Flamingo provides a stronger security guarantee than BBGLR. In Flamingo, an adversary who controls the server and some clients learns a sum that contains inputs from at least a $1 - \delta - \eta$ fraction of clients. In contrast, the malicious protocol in BBGLR leaks several sums: consider a partition of the $|S_t|$ clients, where each partition set has size at least $\alpha \cdot |S_t|$; a malicious adversary in BBGLR learns the sum of each of the partition sets. Concretely, for 5K clients, when both δ and η are 0.2, $\alpha < 0.5$. This means that the adversary learns the sum of two subsets. This follows from Definition 4.1 and Theorem 4.9 in BBGLR [BBG+20].

2.2.4 Cryptographic building blocks

We start by reviewing some standard cryptographic primitives used by BBGLR and Flamingo.

Pseudorandom generators. A PRG is a deterministic function that takes a random seed in $\{0,1\}^\lambda$ and outputs a longer string that is computationally indistinguishable from a random string (λ is the computational security parameter). For simplicity, whenever we use PRG with a seed that is not in $\{0,1\}^\lambda$, we assume that there is a deterministic function that maps the seed to $\{0,1\}^\lambda$ and preserves security. Such mapping is discussed in detail in Section 2.2.9.

Pseudorandom functions. A PRF : $\mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a family of deterministic functions indexed by a key in $\mathcal{K}$ that map an input in $\mathcal{X}$ to an output in $\mathcal{Y}$ in such a way that the indexed function is computationally indistinguishable from a truly random function from $\mathcal{X}$ to $\mathcal{Y}$. We assume a similar deterministic map for inputs as described in the PRG above.

Shamir's secret sharing. An ℓ-out-of-L secret sharing scheme consists of the following two algorithms, *Share* and *Recon*. $\mathit{Share}(s, \ell, L) \to (s_1, \ldots, s_L)$ takes in a secret s, a threshold ℓ, and the number of desired shares L, and outputs L shares $s_1, \ldots, s_L$. *Recon* takes in at least $\ell + 1$ of the shares, and output the secret s; i.e., for a set $U \subseteq [L]$ and $|U| \geq \ell+1$, $\mathit{Recon}(\{s_u\}_{u \in U}) \to s$. Security requires that fewer than $\ell + 1$ shares reveal no information about s.

Diffie-Hellman key exchange. Let $\mathbb{G}$ be a group of order q in which the Decisional Diffie-Hellman (DDH) problem is hard, and g be a generator of $\mathbb{G}$. Alice and Bob can safely establish a shared secret (assuming a passive adversary) as follows. Alice samples a secret $a \xleftarrow{\$} \mathbb{Z}_q$, and sets her public value to $g^a \in \mathbb{G}$. Bob samples his secret $b \xleftarrow{\$} \mathbb{Z}_q$, and sets his public value to $g^b \in \mathbb{G}$. Alice and Bob exchange the public values and raise the other party's value to their secret, i.e., $g^{ab} = (g^a)^b = (g^b)^a$. If DDH is hard, the shared secret g^{ab} is only known to Alice and Bob but no one else.

2.2.5 Limitations of prior work

In this section, we discuss BBGLR [BBG+20], which is the state of the art protocol for a single round secure aggregation in the federated learning setting. We borrow some ideas from this protocol, but design Flamingo quite differently in order to support a full training session.

BBGLR is designed for computing a single sum on the inputs of a set of clients. To apply it to the federated learning setting, we can simply assume that in a given iteration of the training process, there are n clients selected from a large population of size N. We can then run BBGLR on these n clients to compute a sum of their inputs.

The high level idea of BBGLR is for clients to derive pairwise random masks and to add those masks to their input vectors in such a way that when all the masked input vectors across all clients are added, the masks cancel out. It consists of a *setup* phase and a *collection* phase. We first describe a semi-honest version below.

Setup phase. The setup phase consists of three steps: (1) create a database containing public keys of all of the n clients; (2) create an undirected graph where vertices are clients, and each vertex has enough edges to satisfy certain properties; (3) have each client send shares of two secrets to its

neighbors in the graph. We discuss these below.

In the first round, each client $i \in [n]$ generates a secret a_i and sends g^{a_i} to the server, where g^{a_i} represents client i's public key. The server then stores these public keys in a database. Note that the malicious-secure version of BBGLR requires the server to be semi-honest for this particular step, or the existence of a trusted public key infrastructure (PKI).

In the second round, the graph is established as follows. Each client $i \in [n]$ randomly chooses γ other clients in $[n]$ as its neighbors, and tells the server about their choices. After the server collects all the clients' choices, it notifies each client of their neighbors indexes in $[n]$ and public keys. The neighbors of client i, denoted as $A(i)$, are those corresponding to vertices that have an edge with i (i.e., i chose them or they chose i).

Finally, each client i uses Shamir's secret sharing to share a_i and an additional random value m_i to its neighbors $A(i)$ (let the threshold be $\ell < |A(i)|$), where the shares are end-to-end encrypted with a secure authenticated encryption scheme and sent via the server (§2.2.3).

Collection phase. Client i sends the following masked vector to the server:

$$Vec_i = \mathbf{x}_i + \underbrace{\sum_{j \in A(i), i<j} \text{PRG}(r_{i,j}) - \sum_{j \in A(i), i>j} \text{PRG}(r_{i,j})}_{\text{pairwise mask}} + \underbrace{\text{PRG}(m_i)}_{\text{individual mask}},$$

where $r_{i,j} = g^{a_i a_j}$, which can be computed by client i since it has the secret a_i and j's public key, g^{a_j}. (These are essentially Diffie-Hellman key exchanges between a client and its neighbors.) Here we view the output of the PRG as a vector of integers instead of a binary string. Also, we will write the pairwise mask term as $\sum_{j \in A(i)} \pm\text{PRG}(r_{i,j})$ for ease of notation.

As we mentioned earlier (§2.2.3), clients may drop out due to unstable network conditions, power loss, etc. This means that the server may not receive some of the masked vectors within an acceptable time period. Once the server times out, the server labels the clients whose vectors have been received as "online"; the rest are labeled "offline". The server shares this information with all the n clients. The server then sums up all of the received vectors, which yields a vector that contains the masked

sum. To recover the correct sum, the server needs a way to remove the masks. It does so by requesting for each *offline* client i, the shares of a_i from i's neighbors; and for each *online* client j, the shares of m_j from j's neighbors. These shares allow the server to reconstruct either the pairwise mask or the individual mask for each client. As long as there are more than ℓ neighbors that send the requested shares, the server can successfully remove the masks and obtain the sum. This gives the dropout resilience property of BBGLR.

One might wonder the reason for having the individual mask m_i, since the pairwise mask already masks the input. To see the necessity of having m_i, assume that it is not added, i.e., $Vec_i = \mathbf{x}_i + \sum \pm \text{PRG}(r_{ij})$. Suppose client i's message is sent but not received on time. Thus, the server reconstructs i's pairwise mask $\sum \pm \text{PRG}(r_{ij})$. Then, i's vector Vec_i arrives at the server. The server can then subtract the pairwise mask from Vec_i to learn $\mathbf{x}_i$. The individual mask m_i prevents this.

Preventing attacks in fault recovery. The above protocol only works in the semi-honest setting. There are two major attacks that a malicious adversary can perform. First, a malicious server can give inconsistent dropout information to honest clients and recover both the pairwise and individual masks. For example, suppose client i has neighbors $j_1, \ldots, j_\gamma$, and a malicious server lies to the neighbors of $j_1, \ldots, j_\gamma$ that $j_1, \ldots, j_\gamma$ have dropped out (when they actually have not). In response, their neighbors, including i, will provide the server with the information it needs to reconstruct $a_{j_1}, \ldots, a_{j_\gamma}$, thereby deriving all the pairwise secrets $r_{i,j_1}, \ldots, r_{i,j_\gamma}$. At the same time, the server can tell $j_1, \ldots, j_\gamma$ that i was online and request the shares of m_i. This gives the server both the pairwise mask and the individual mask of client i, violating i's privacy. To prevent this, BBGLR has a consistency check step performed among all neighbors of each client to reach an agreement on which nodes actually dropped out. In this case, i would have learned that none of its neighbors dropped out and would have refused to give the shares of their pairwise mask.

Second, malicious clients can submit a share that is different than the share that they received from their neighbors. This could lead to reconstruction failure at the server, or to the server deriving a completely different secret. BBGLR fixes the latter issue by having the clients hash their secrets and send these hashes to the server when they send their input vectors; however, reconstruction

could still fail because of an insufficient threshold in error correction[1].

In sum, the full protocol of BBGLR that withstands a malicious adversary (assuming a PKI or a trusted server during setup) has six rounds in total: three rounds for the setup and three rounds for computing the sum.

Using BBGLR for federated learning

BBGLR works well for *one* iteration of training, but when many iterations are required, several issues arise. First, in federated learning the set of clients chosen to participate change over iterations, so a new graph needs to be derived and new secrets must be shared. Even if the graph stays the same, the server cannot reuse the secrets from the setup in previous aggregation as the masks are in fact one-time pads that cannot be applied again. This means that we must run the setup phase for each aggregation, which incurs a high latency since the setup contains three rounds involving all the clients.

Moreover, BBGLR's threat model does not naturally extend to multiple executions of aggregation. It either needs a semi-honest server or a PKI during the first round of the protocol. If we assume the former, then this means the adversary has to be semi-honest during the exact time of setup in each training iteration, which is practically impossible to guarantee. If we use a PKI, none of the keys can be reused (for the above reasons); as a result, all of the keys in the PKI need to be updated for each iteration, which is costly.

2.2.6 High-level ideas

Flamingo supports continuous execution of multiple aggregations without redoing the setup for each aggregation and withstands a malicious adversary throughout. The assumptions required are: (1) in the setup, all parties are provided with the same random seed from a trusted source (e.g., a distributed randomness beacon [DKIR21]); and (2) a PKI (e.g., a key transparency log [CDGM19, MBB+15, LGG+22, TBP+19, TFZ+22, TKPS22, HHK+21]). Below we describe the high-level ideas

[1]To apply a standard error correction algorithm such as Berlekamp-Welch in this setting, the polynomial degree should be at most $\gamma/3$. Definition 4.2 in BBGLR implies that the polynomial degree may be larger than required for error correction.

underpinning Flamingo and then we give the full protocol (§2.2.7 and §2.2.7).

Flamingo has three key ideas:

(1) Lightweight dropout-resilience. Instead of asking clients to secret share a_i and m_i for their masks with all of their neighbors, we let each client encrypt—in a special way—the PRG seeds of their pairwise and individual masks, append the resulting ciphertexts to their masked input vectors, and submit them to the server in a single round. Then, with the help of a special set of L clients that we call *decryptors*, the server can decrypt one of the two seeds associated with each masked input vector, but not both. In effect, this achieves a similar fault-tolerance property as BBGLR (§2.2.5), but with a different mechanism.

The crucial building block enabling this new protocol is *threshold decryption* [GHKR08, DF89, SG02], in which clients can encrypt data with a system-wide known public key PK, in such a way that the resulting ciphertexts can only be decrypted with a secret key SK that is secret shared among decryptors in Flamingo. Not only does this mechanism hide the full secret key from every party in the system, but the decryptors can decrypt a ciphertext without ever having to interact with each other. Specifically, the server in Flamingo sends the ciphertext (pealed from the submitted vector) to each of the decryptors, obtains back some threshold ℓ out of L responses, and locally combines the ℓ responses which produce the corresponding plaintext. Our instantiation of threshold decryption is based on the ElGamal cryptosystem and Shamir's secret sharing; we describe it in Section 2.2.9. Technically one can also instantiate the threshold decryption with other cryptosystems, but we choose ElGamal because it enables non-interactive threshold decryption, simple distributed key generation, and efficient proof of decryption (for malicious security, §2.2.7).

One key technical challenge that we had to overcome when designing this protocol is figuring out how to secret share the key SK among the decryptors. To our knowledge, existing efficient distributed key generation (DKG) protocols [Ped91, CS04, GJKR06] assume a broadcast channel or reliable point-to-point channels, whereas our communication model is that of a star topology where all messages are proxied by a potential adversary (controlling the server) that can drop them. There

are also asynchronous DKG protocols [KKMS20, DYX+22, AJM+23], but standard asynchronous communication model assumes eventual delivery of messages which is not the case in our setting. In fact, we can relax the guarantees of DKG and Section 2.2.7 gives an extension of a discrete-log-based DKG protocol [GJKR06] (since we target ElGamal threshold decryption) that works in the star-topology communication model.

In sum, the above approach gives a dropout-resilient protocol for a single summation with two rounds: first, each client sends their masked vector and the ciphertexts of the PRG seeds; second, the server uses distributed decryption to recover the seeds (and the masks) for dropout clients (we discuss how to ensure that decryptors agree on which of the two seeds to decrypt in §2.2.7). This design improves the run time over BBGLR by eliminating the need to involve all the clients to remove the masks—the server only needs to wait until it has collected enough shares from the decryptors, instead of waiting for almost all the shares to arrive. Furthermore, the communication overhead of appending several small ciphertexts (64 bytes each) to a large input vector (hundreds of KBs) is minimal.

(2) Reusable secrets. Flamingo's objective is to get rid of the setup phase for each aggregation. Before we discuss Flamingo's approach, consider what would happen if we were to naively run the setup phase in BBGLR once, followed by running the collection procedure multiple times. First, we are immediately limited to performing all of the aggregation tasks on the same set of clients, since BBGLR establishes the graph of neighbors during the setup phase. This is problematic since federated learning often chooses different sets of clients for each round of aggregation (§2.2.3). Second, clients' inputs are exposed. To see why, suppose that client i drops out in iteration 1 but not in 2. In iteration 1, the server reconstructs $r_{i,j}$ for $j \in A(i)$ to unmask the sum. In iteration 2, client i sends $\mathbf{x}_i + \sum_{j \in A(i)} \pm \text{PRG}(r_{i,j}) + \text{PRG}(m_i)$ and the server reconstructs m_i by asking i's neighbors for their shares. Since all the $r_{i,j}$ are reconstructed in iteration 1 and are reused in iteration 2, the server can derive both masks.

The above example shows that the seeds should be new and independent in each iteration of aggregation. We accomplish this with a simple solution that adds a level of indirection. Flamingo treats $r_{i,j}$

as a long-term secret and lets the clients apply a PRF to generate a new seed for each pairwise mask. Specifically, clients i computes the PRG seed for pairwise mask in round t as $h_{i,j,t} := \text{PRF}(r_{i,j}, t)$ for all $j \in A(i)$. Note that client j will compute the same $h_{i,j,t}$ as it agrees with i on $r_{i,j}$. In addition, each client also generates a fresh seed $m_{i,t}$ for the individual mask in iteration t. Consequently, combined with idea (1), each client uses PK to encrypt the per-round seeds, $\{h_{i,j,t}\}_{j \in A(i)}$ and $m_{i,t}$. Then, the server recovers one of the two for each client. We later describe an optimization where clients do not encrypt $m_{i,t}$ with PK (§2.2.7).

A nice property of $r_{i,j}$ being a long-term secret is that Flamingo can avoid performing all the Diffie-Hellman key exchanges between graph neighbors (proxied through the server). Flamingo relies instead on an external PKI or a verifiable public key directory such as CONIKS [MBB+15] and its successors (which are a common building block for bootstrapping end-to-end encrypted systems).

We note that this simple technique cannot be applied to BBGLR to obtain a secure multi-round protocol. It is possible in Flamingo precisely because clients encrypt their per-round seeds for pairwise masks directly so the server never needs to compute these seeds from the long-term pairwise secrets. In contrast, in BBGLR, clients derive pairwise secrets (g^{a_i,a_j}) during the setup phase. When client i drops out, the server collects enough shares to reconstruct a_i and compute the pairwise secrets, g^{a_i,a_j}, for all online neighbors j of client i. Even if we use a PRF here, the server already has the pairwise secret; so it can run the PRF for any iteration and conduct the attacks described earlier.

(3) Per-iteration graphs. BBGLR uses a sparse graph instead of a fully-connected graph for efficiency reasons (otherwise each client would need to secret share its seeds with every other client). In federated learning, however, establishing sparse graphs requires per-iteration generation since the set S_t changes in each iteration (some clients are chosen again, some are new [LZMC21]). A naive way to address this is to let all clients in $[N]$ establish a big graph G with N nodes in the setup phase: each client in $[N]$ sends its choice of γ neighbors to the server, and the server sends to each client the corresponding neighbors. Then, in each iteration t, the corresponding subgraph G_t consists of clients in S_t and the edges among clients in S_t.

```
Parameters: ϵ.                                          // the probability that an edge is added
function ChooseSet(v, t, n_t, N)
    S_t ← ∅.
    v*_t := PRF(v, t).
    while |S_t| < n_t do
        Parse log N bits from PRG(v*_t) as i, add i to S_t.
    Output S_t.
function GenGraph(v, t, S_t)
    G_t ← n_t × n_t empty matrix; ρ ← log(1/ϵ).
    for i ∈ S_t, j ∈ S_t do
        Let v' be the first ρ bits of PRF(v, (i, j)).
        if v' = 0^ρ then set G_t(i, j) := 1
    Output G_t.
function FindNeighbors(v, S_t, i)
    A_t(i) ← ∅; ρ ← log(1/ϵ).
    for j ∈ S_t do
        Let v' be the first ρ bits of PRF(v, (i, j)).
        if v' = 0^ρ then add j to A_t(i).
    for j ∈ S_t do
        Let v' be the first ρ bits of PRF(v, (j, i)).
        if v' = 0^ρ then add j to A_t(i).
    Output A_t(i).
```

Figure 2.1: Pseudocode for generating graph G_t in round t.

However, this solution is unsatisfactory. If one uses a small γ (e.g., $\log N$), G_t might not be connected and might even have isolated nodes (leaking a client's input vector since it has no pairwise masks); if one uses a large γ (e.g., the extreme case being N), G_t will not be sparse and the communication cost for the server will be high (e.g., $O(N^2)$).

Flamingo introduces a new approach for establishing the graph with a communication cost independent of γ. The graph for each iteration t is generated by a random string $v \in \{0,1\}^\lambda$ known to all participants (obtained from a randomness beacon or a trusted setup). Figure 2.1 lists the procedure. $\textsc{ChooseSet}(v, t, n_t, N)$ determines the set of clients involved in iteration t, namely $S_t \subseteq [N]$ with size n_t. The server computes $G_t \leftarrow \textsc{GenGraph}(v, t, S_t)$ as the graph in iteration t among clients in S_t. A client $i \in S_t$ can find its neighbors in G_t without materializing the whole graph using $\textsc{FindNeighbors}(v, S_t, i)$. In this way, the neighbors of i can be locally generated. We choose a proper ϵ such that in each iteration, the graph is connected with high probability (details in §2.2.8).

Figure 2.2: Workflow of Flamingo. The server first does a setup for all clients in the system. In each round t of training, the server securely aggregates the masked input vectors in the report step; in the cross-check and reconstruction steps, the server communicates with a small set of randomly chosen clients who serve as decryptors. The decryptors are chosen independently from the set S_t that provides inputs in a given round. Every R rounds, the decryptors switch and the old decryptors transfers shares of SK to new decryptors.

The above ideas taken together eliminate the need for per-iteration setup, which improves the overall run time of multi-iteration aggregation over BBGLR. Figure 2.2 depicts the overall protocol, and the next sections describe each part.

2.2.7 Full protocol description

Before giving our protocol, we need to specify what types of keys the PKI needs to store. The keys depend on the cryptographic primitives that we use (signature schemes, symmetric encryption and ElGamal encryption); for ease of reading, we formally give these primitives in Appendix A.2.1.

The PKI stores three types of keys for all clients in $[N]$:

- g^{a_i} of client i for its secret a_i; this is for client j to derive the pairwise secret $r_{i,j}$ with client i by computing $(g^{a_j})^{a_i}$.
- g^{b_i} of client i for deriving a symmetric encryption key $k_{i,j}$ for an authenticated encryption scheme SymAuthEnc (Definition A.2.3); this scheme is used when a client sends messages to another client via the server. Later when we say client i sends a message to client j via the server in the protocol, we implicitly assume the messages are encrypted using $k_{i,j}$.
- pk_i of client i for verifying i's signature on messages signed by sk_i.

Setup phase

The setup phase consists of two parts: (1) distributing a random seed $v \in \{0,1\}^\lambda$ to all participants, and (2) selecting a random subset of clients as decryptors and distribute the shares of the secret

key of an asymmetric encryption scheme AsymEnc. In our context, AsymEnc is the ElGamal cryptosystem's encryption function (Definition A.2.2).

As we mentioned earlier, the first part can be done through a trusted source of randomness, or by leveraging a randomness beacon that is already deployed, such as Cloudflare's [clo]. The second part can be done by selecting a set of L clients as decryptors, $\mathcal{D}$, using the random seed v (ChooseSet), and then running a DKG protocol among them. We use a discrete-log based DKG protocol [GJKR06] (which we call GJKR-DKG) since it is compatible with the ElGamal cryptosystem. However, this DKG does not work under our communication model and requires some changes and relaxations, as we discuss next.

DKG with an untrusted proxy. The correctness and security of the GJKR-DKG protocol relies on a secure broadcast channel. Our communication model does not have such a channel, since the server can tamper, replay or drop messages. Below we give the high-level ideas of how we modify GJKR-DKG and Appendix A.2.2 gives the full protocol.

We begin with briefly describing the GJKR-DKG protocol. It has a threshold of 1/2, which means that at most half of the participants can be dishonest; the remaining must perform the following steps correctly: (1) Each party i generates a random value s_i and acts as a dealer to distribute the shares of s_i (party j gets $s_{i,j}$). (2) Each party j verifies the received shares (we defer how the verification is done to Appendix A.2.2). If the share from the party i fails the verification, j broadcasts a complaint against party i. (3) Party i broadcasts, for each complaint from party j, the $s_{i,j}$ for verification. (4) Each party disqualifies those parties that fail the verification; the rest of the parties form a set QUAL. Then each party sums up the shares from QUAL to derive a share of the secret key.

Given our communication model, it appears hard to guarantee the standard DKG correctness property, which states that if there are enough honest parties, at the end of the protocol the honest parties hold valid shares of a unique secret key. Instead, we relax this correctness property by allowing honest parties to instead abort if the server who is proxying the messages acts maliciously.

We modify GJKR-DKG in the following ways. First, we assume the threshold of dishonest participants is 1/3. Second, all of the messages are signed; honest parties abort if they do not receive the prescribed messages. Third, we add another step before each client decides on the eventual set QUAL: all parties sign their QUAL set and send it to the server; the server sends the signed QUALs to all the parties. Each party then checks whether it receives $2\ell + 1$ or more valid signed QUAL sets that are the same. If so, then the QUAL set defines a secret key; otherwise the party aborts. We give the detailed algorithms and the corresponding proofs in Appendix A.2.2. Note that the relaxation from GJKR-DKG is that we allow parties to abort (so no secret key is shared at the end), and this is reasonable particularly in the federated learning setting because the server will not get the result if it misbehaves.

Decryptors run DKG. At the end of our DKG, a *subset* of the selected decryptors will hold the shares of the secret key SK. The generated PK is signed by the decryptors and sent to all of the N clients by the server; the signing prevents the server from distributing different public keys or distributing a public key generated from a set of malicious clients. Each client checks if it received $2\ell + 1$ valid signed PKs from the set of decryptors determined by the random seed (from beacon); if not, the client aborts. In Appendix A.2, we provide the pseudocode for the entire setup protocol Π_{setup} (Fig. A.1).

Collection phase

The collection phase consists of T iterations of aggregation; each iteration t has three steps: report, cross-check, and reconstruction. Each step involves a communication round between the server and (some of the) clients. Below we describe each step and we defer the full protocol Π_{sum} to Figure A.4 in Appendix A.2. The cryptographic primitives we use here (SymAuthEnc and AsymEnc) are formally given in Appendix A.2.1.

Report. In iteration t, the server uses the random value v (obtained from the setup) to select the set of clients $S_t \subseteq [N]$ of size n_t by running $\textsc{ChooseSet}(v, t, n_t, N)$. It then establishes the graph $G_t \leftarrow \textsc{GenGraph}(v, t, S_t)$ as specified in Figure 2.1. We denote the neighbors of i as $A_t(i) \subseteq S_t$. The server asks each client $i \in S_t$ to send a message consisting of the following three things:

1. $Vec_i = \mathbf{x}_{i,t} + \sum_{j \in A_t(i)} \pm \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{i,t})$, where $h_{i,j,t}$ is computed as $\text{PRF}(r_{i,j}, t)$ and $r_{i,j}$ is derived from the key directory by computing $(g^{a_j})^{a_i}$; $m_{i,t}$ is freshly generated in iteration t by client i.

2. L symmetric ciphertexts: $\text{SymAuthEnc}(k_{i,u}, m_{i,u,t})$ for all $u \in \mathcal{D}$, where $m_{i,u,t}$ is the share of $m_{i,t}$ meant for u (i.e., $\mathit{Share}(m_{i,t}, \ell, L) \to \{m_{i,u,t}\}_{u \in \mathcal{D}}$), and $k_{i,u}$ is the symmetric encryption key shared between client i and decryptor u (they can derive $k_{i,u}$ from the PKI);

3. $|A_t(i)|$ ElGamal ciphertexts: $\text{AsymEnc}(PK, h_{i,j,t})$ for all $j \in A_t(i)$.

The above way of using symmetric encryption for individual masks and public-key encryption for pairwise masks is for balancing computation and communication in practice. Technically, one can also encrypt the shares of $h_{i,j,t}$ with symmetric authenticated encryption as well (eliminating public-key operations), but it increases client communication—for each client, the number of ciphertexts appended to the vector is $|A(i)| \cdot L$. This is, for example, 1,600 when L and $|A(i)|$ are both 40. On the other hand, if one encrypts both the pairwise and individual masks using only public-key encryption, then the number of expensive public key operations for reconstructing secrets is proportional to n_t; whereas it is only proportional to the number of dropouts in our proposed approach. In practice, the number of dropouts is much smaller than n_t, hence the savings.

Cross-check. The server needs to recover $m_{i,t}$ for online clients, and $h_{i,j,t}$ for clients that drop out. To do so, the server labels the clients as "offline" or "online" and asks the decryptors to recover the corresponding masks. For BBGLR, we described how this step involves most clients during the fault recovery process and highlighted an issue where a malicious server can send inconsistent labels to clients and recover both the pairwise mask and individual mask for some target client (§2.2.5). Flamingo also needs to handle this type of attack (the server tells some of honest decryptors to decrypt $m_{i,t}$ and other honest decryptors to decrypt $h_{i,j,t}$, and utilizes the malicious decryptors to reconstruct both), but it only needs to involve decryptors. In detail, each decryptor signs the online/offline labels of the n_t clients (each client can only be labeled either offline or online), and sends them to the other decryptors (via the server). Each decryptor checks it received $2L/3$ or more

valid signed labels (recall from §2.2.3 that $\delta_D + \eta_D < 1/3$). If so, each decryptor further checks that:

1. The number of online clients is at least $(1-\delta)n_t$;
2. All the online clients in the graph are connected;
3. Each online client i has at least k online neighbors[2], such that $\eta^k < 2^{-\kappa}$ (η and κ are defined as in §2.2.3).

If any of the above checks fail, the decryptor aborts. This step ensures either all the honest decryptors agree on a valid offline/online label assignment and consequently the server gets the result, or the honest decryptors abort and the server gets nothing.

Reconstruction. The server collects all the ciphertexts to be decrypted: the ciphertexts of $m_{i,u,t}$ (symmetric encryption) for the online clients, and the ciphertexts of $h_{i,j,t}$ (public-key encryption) for the offline clients. Then the server sends the ciphertexts to all the decryptors who perform either a symmetric decryption or the threshold ElGamal decryption according to their agreed-upon labels.

The choice of using decryptors to check the graph and reconstruct all the secrets is based on an important observation in federated learning: the number of clients involved in one iteration, n_t, is much smaller than the input vector length [KMA+21]. Therefore, the asymptotic costs at a decryptor (which are proportional to n_t) are actually smaller than the size of an input weight vector.

Malicious labeling across iterations

The server, controlled by a malicious adversary, can ask for the decryption of $h_{i,j,t}$ in iteration t, and then in some other iteration t', the server can ask for the decryption of $m_{i,t}$ (but not $m_{i,t'}$, if the server does not care about obtaining a result in iteration t'). This allows the server to recover $\mathbf{x}_{i,t}$ in the clear. To prevent this attack, honest decryptors need to know the iteration for which

[2]This can be efficiently done in an inverse way of checking how many offline neighbors that each online client has, assuming dropout rate is small.

a ciphertext is sent. For symmetric ciphertext, the client appends the iteration number t to the plaintext (e.g., $m_{i,u,t}||t$) and uses authenticated encryption; for public-key ciphertexts, the client appends t to the ciphertext c and signs the tuple (c, t) (the verification key is in the PKI). Note that a malicious adversary can still fool enough honest decryptors into thinking it is iteration t while it is in fact t'. To prevent this, decryptors also include the iteration number in the online/offline labels and sign them. The cross-check (§2.2.7) guarantees that the decryptors agree on the iteration number.

Load balancing across decryptors

In each summation, a client who is not a decryptor only sends a single vector. This is nearly optimal since even if the aggregation is non-private the client has to send the vector (but without the additional small ciphertexts). The decryptors, however, have additional computation and communication in order to help with the result reconstruction. This causes a load imbalance in the system and could be unfair since a client selected to be a decryptor has to do more work than regular clients.

In Flamingo, the decryptor responsibility shifts across time. Every R iterations, the current decryptors transfer their shares of SK to a new set of randomly selected clients who serve as the new decryptors. To ensure security, the shares of SK have to be modified in a particular way during the transition, as otherwise the adversary may control some malicious decryptors before the transition and some malicious decryptors after the transition, and thus may obtain enough shares to reconstruct SK. We address this by relying on prior proactive secret sharing techniques [HJKY95, GHKR08, KMZ+19]; they additionally enable Flamingo to change the number of decryptors and the threshold as needed. In Appendix A.2.4, we provide details of the transition protocol used in Flamingo.

A final clarification is that decryptors who dropped out (e.g., due to power loss) at one iteration can come back later and participate in another iteration (e.g., when power is resumed). The decryption always succeeds since we require that less than 1/3 deryptors are dropped out or malicious at any round (§2.2.8). The secret key transition is purely for system load balancing—neither dropout

resilience nor security relies on the parameter R.

Considerations in federated learning

A recent work [PFA22] describes an attack in the composition of federated learning and secure aggregation. The idea is that the server can elude secure aggregation by sending clients inconsistent models. For example, the server sends to client 1 model M_1, to client 2 model M_2, and to client 3 model M_3. Each of the clients then runs the local training on the provided model. The server chooses the models it sends to clients 1 and 2 in a special way such that after clients 1 and 2 train their local models, their local weights will cancel out when added. Consequently, the server will get the model weights of client 3. The proposed defense, which works in our context without incurring any overhead, is for clients to append the hash of the model they receive in a given iteration to their PRG seed for that iteration: $\text{PRG}(h_{i,j,t}||Hash(M))$, where M is the model received from the server. If all the clients receive the same model, the pairwise masks cancel out; otherwise, they do not.

2.2.8 Security analysis and parameter selection

The parameters of Flamingo include:

- System parameters N, T and the number of clients n_t chosen in iteration $t \in [T]$;
- Threat model parameters δ_D, δ, η which are given, and η_{S_t}, η_D which depend on η (their relation is summarized in Section 2.2.3 and fully derived in Appendix A.1).
- Security parameter κ, and the parameters that relates to security: graph generation parameter ϵ, and the number of selected decryptors L.

We discuss these parameters in detail below and state our formal lemmas with respect to them.

Security of setup phase

Let δ_D upper bound the fraction of decryptors that drop out during the setup phase; note that in Section 2.2.3 we let δ_D upper bound the dropouts in one aggregation and for simplicity here we use the same notation. Flamingo's DKG requires that $\delta_D + \eta_D < 1/3$. Note that η_D in fact depends

Functionality $\mathcal{F}_{\text{setup}}$

Parties: clients $1, \ldots, N$ and a server.

- $\mathcal{F}_{\text{setup}}$ samples $v \xleftarrow{\$} \{0,1\}^{\lambda}$.
- $\mathcal{F}_{\text{setup}}$ samples a secret key and public key pair (SK, PK).

  // When the public-key cryptosystem is instantiated by ElGamal, then SK is $s \xleftarrow{\$} \mathbb{Z}_q$ and $PK = g^s$.
- $\mathcal{F}_{\text{setup}}$ asks the adversary $\mathcal{A}$ whether it should continue or not. If $\mathcal{A}$ replies with `abort`, $\mathcal{F}_{\text{setup}}$ sends `abort` to all honest parties; if $\mathcal{A}$ replies with `continue`, $\mathcal{F}_{\text{setup}}$ sends v and PK to all the parties.

Figure 2.3: Ideal functionality for the setup phase.

on η, L and N, but we will give the theorems using η_D and discuss how to choose L to guarantee a desired η_D in Appendix A.3.

Theorem 2.2.1 (Security of setup phase)**.** *Assume that a PKI and a trusted source of randomness exist, and that the DDH assumption holds. Let the dropout rate of decryptors in the setup phase be bounded by δ_D. If $\delta_D + \eta_D < 1/3$, then under the communication model defined in Section 2.2.3, protocol Π_{setup} (Fig. A.1) securely realizes functionality $\mathcal{F}_{setup}$ (Fig. 2.3) in the presence of a malicious adversary controlling the server and η fraction of the N clients.*

Security of collection phase

First, we need to guarantee that each graph G_t, even *after* removing the vertices corresponding to the $\delta + \eta$ fraction of dropout and malicious clients, is still connected. This imposes a requirement on ϵ, which we state in Lemma 2.2.2. For simplicity, we omit the exact formula for the lower bound of ϵ and defer the details to Appendix A.3.

Lemma 2.2.2 (Graph connectivity)**.** *Given a security parameter κ, and threat model parameters δ, η (§2.2.3). Let G be a random graph $G(n, \epsilon)$. Let $\mathcal{C}, \mathcal{O}$ be two random subsets of nodes in G where $|\mathcal{O}| \leq \delta n$ and $|\mathcal{C}| \leq \eta n$ ($\mathcal{O}$ stands for dropout set and $\mathcal{C}$ stands for malicious set). Let $\tilde{G}$ be the graph with nodes in $\mathcal{C}$ and $\mathcal{O}$ and the associated edges removed from G. There exists ϵ^* such that for all $\epsilon \geq \epsilon^*$, $\tilde{G}$ is connected except with probability $2^{-\kappa}$.*

Secondly, we require $2\delta_D + \eta_D < 1/3$ to ensure that all online honest decryptors reach an agreement

Functionality $\mathcal{F}_{\text{mal}}$

Parties: clients $1, \ldots, N$ and a server.

Parameters: corrupted rate η, dropout rate δ, number of per-iteration participating clients n.

- $\mathcal{F}_{\text{mal}}$ receives from a malicious adversary $\mathcal{A}$ a set of corrupted parties, denoted as $\mathcal{C} \subset [N]$, where $|\mathcal{C}|/N \leq \eta$.
- For each iteration $t \in [T]$:
  1. $\mathcal{F}_{\text{mal}}$ receives a sized-n random subset $S_t \subset [N]$ and a dropout set $\mathcal{O}_t \subset S_t$, where $|\mathcal{O}_t|/|S_t| \leq \delta$, and $|\mathcal{C}|/|S_t| \leq \eta$, and inputs $\mathbf{x}_{i,t}$ from client $i \in S_t \backslash (\mathcal{O}_t \cup \mathcal{C})$.
  2. $\mathcal{F}_{\text{mal}}$ sends S_t and $\mathcal{O}_t$ to $\mathcal{A}$, and asks $\mathcal{A}$ for a set M_t: if $\mathcal{A}$ replies with $M_t \subseteq S_t \backslash \mathcal{O}_t$ such that $|M_t|/|S_t| \geq 1 - \delta$, then $\mathcal{F}_{\text{mal}}$ computes $\mathbf{y}_t = \sum_{i \in M_t \backslash \mathcal{C}} \mathbf{x}_{i,t}$ and continues; otherwise $\mathcal{F}_{\text{mal}}$ sends `abort` to all the honest parties.
  3. Depending on whether the server is corrupted by $\mathcal{A}$:
     - If the server is corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{mal}}$ outputs $\mathbf{y}_t$ to all the parties corrupted by $\mathcal{A}$.
     - If the server is not corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{mal}}$ asks $\mathcal{A}$ for a shift $\mathbf{a}_t$ and outputs $\mathbf{y}_t + \mathbf{a}_t$ to the server.

Figure 2.4: Ideal functionality for Flamingo protocol.

in the cross-check step and the reconstruction is successful. Note that the decryptors in the setup phase who dropped out (δ_D fraction) will not have the share of SK; while the clients who drop out during a complete iteration (another δ_D fraction) in the collection phase can come back at another iteration, hence we have the above inequality.

Main theorems

The full protocol, denoted as Φ_T, is the sequential execution of Π_{setup} (Fig. A.1) followed by a T-iteration Π_{sum} (Fig. A.4). We now give formal statements for the properties of Flamingo, and defer the proof to Appendix A.4. Note that as we see from the ideal functionality $\mathcal{F}_{\text{mal}}$ (Fig. 2.4), when the server is corrupted, the sum result in iteration t is not determined by the actual dropout set $\mathcal{O}_t$, but instead a set M_t chosen by the adversary (see details in Appendix A.4.5).

Theorem 2.2.3 (Dropout resilience of Φ_T)**.** *Let $\delta, \delta_D, \eta, \eta_D$ be threat model parameters as defined (§2.2.8,§2.2.8). If $2\delta_D + \eta_D < 1/3$, then protocol Φ_T satisfies dropout resilience: when all parties follow the protocol Φ_T, for every iteration $t \in [T]$, and given a set of dropout clients $\mathcal{O}_t$ in the report step where $|\mathcal{O}_t|/|S_t| \leq \delta$, protocol Φ_T terminates and outputs $\sum_{i \in S_t \backslash \mathcal{O}} \mathbf{x}_{i,t}$, except probability $2^{-\kappa}$.*

		BBGLR			Flamingo		
Phase		**Steps**	**Server**	**Client**	**Steps**	**Server**	**Client**
Setup		—	—	—	4	$O(L^3)$	$O(L^2)$
T sums	Iteration setup	$3T$	$O(TAn_t)$	$O(TA)$	—	—	—
	Collection	$3T$	$O(Tn_t(d+A))$	$O(T(d+A))$	$3T$	$O(Tn_t(d+L+A))$	Regular client: $O(T(d+A))$ Decryptors: $O(T(L+\delta An_t+(1-\delta)n_t))$

Figure 2.5: Communication complexity and number of steps (client-server round trips) of Flamingo and BBGLR for T iterations of aggregation. N is the total number of clients and n_t is the number of clients chosen to participate in iteration t. The number of decryptors is L, and the dropout rate of clients in S_t is δ. Let A be the upper bound on the number of neighbors of a client, and let d be the dimension of client's input vector.

Theorem 2.2.4 (Security of Φ_T)**.** *Let the security parameter be κ. Let $\delta, \delta_D, \eta, \eta_D$ be threat model parameters as defined (§2.2.8,§2.2.8). Let ϵ be the graph generation parameter (Fig. 2.1). Let N be the total number of clients and n be the number of clients for summation in each iteration. Assuming the existence of a PKI, a trusted source of initial randomness, a PRG, a PRF, an asymmetric encryption AsymEnc, a symmetric authenticated encryption SymAuthEnc, and a signature scheme, if $2\delta_D + \eta_D < 1/3$ and $\epsilon \geq \epsilon^*(\kappa)$ (Lemma 2.2.2), then under the communication model defined in Section 2.2.3, protocol Φ_T securely realizes the ideal functionality $\mathcal{F}_{mal}$ given in Figure 2.4 in the presence of a static malicious adversary controlling η fraction of N clients (and the server),[3] except with probability at most $Tn \cdot 2^{-\kappa+1}$.*

The final complication is how to choose L to ensure $2\delta_D + \eta_D < 1/3$ holds; note that η_D depends on η and L. One can choose L to be N but it does not give an efficient protocol; on the other hand, choosing a small L may result in all the decryptors being malicious. In Appendix A.3, we give a lower bound of L to ensure a desired η_D (w.h.p.), given N, η, and δ_D.

2.2.9 Implementation details

We implement Flamingo in 1.7K lines and BBGLR in 1.1K lines of Python. We choose Python to facilitate integration with the machine learning training pipeline. For PRG, we use AES in counter mode, for authenticated encryption we use AES-GCM, and signatures use ECDSA over curve P-256. Our code is available at `https://github.com/eniac/flamingo`.

[3]We assume that in each iteration, the corrupted fraction of n clients is also η; see Section 2.2.3.

Distributed decryption. We build the distributed decryption scheme discussed in Section 2.3.4 as follows. We use ElGamal encryption to instantiate the asymmetric encryption. It consists of three algorithms (AsymGen, AsymEnc, AsymDec). AsymGen outputs a secret and public key pair $SK \in_R \mathbb{Z}_q$ and $PK := g^{SK} \in \mathbb{G}$. AsymEnc takes in PK and plaintext $h \in \mathbb{G}$, and outputs ciphertext $(c_0, c_1) := (g^w, h \cdot PK^w)$, where $w \in_R \mathbb{Z}_q$ is the encryption randomness. AsymDec takes in SK and ciphertext (c_0, c_1) and outputs $h = (c_0^{SK})^{-1} \cdot c_1$.

In threshold decryption [GHKR08, DF89, SG02], the secret key SK is shared among L parties such that each party $u \in [L]$ holds a share s_u, but no single entity knows SK, i.e., $(s_1, \ldots, s_L) \leftarrow$ *Share*(SK, ℓ, L). Suppose Alice wants to decrypt the ciphertext (c_0, c_1) using the secret-shared SK. To do so, Alice sends c_0 to each party in $[L]$, and gets back $c_0^{s_u}$ for $u \in U \subseteq [L]$. If $|U| > \ell$, Alice can compute from U a set of combination coefficients $\{\beta_u\}_{u \in U}$, and

$$c_0^{SK} = \prod_{u \in U} (c_0^{s_u})^{\beta_u}.$$

Given c_0^{SK}, Alice can get the plaintext $h = (c_0^{SK})^{-1} \cdot c_1$. Three crucial aspects of this protocol are that: (1) SK is never reconstructed; (2) the decryption is dropout resilient (Alice can obtain h as long as more than ℓ parties respond); (3) it is non-interactive: Alice communicates with each party exactly once.

We implement ElGamal over elliptic curve group $\mathbb{G}$ and we use curve P-256. To map the output of PRF$(r_{i,j}, t)$, which is a binary string, to $\mathbb{G}$, we first hash it to an element in the field of the curve using the *hash-to-field* algorithm from the IETF draft [HSS$^+$21, §5]. We then use the SSWU algorithm [BCI$^+$10, WB19] to map the field element to a curve point $P \in \mathbb{G}$. A client will encrypt P with ElGamal, and then hash P with SHA256 to obtain $h_{i,j,t}$—the input to the pairwise mask's PRG. When the server decrypts the ElGamal ciphertexts and obtains P, it uses SHA256 on P to obtain the same value of $h_{i,j,t}$.

Optimizations. In Flamingo's reconstruction step, we let the server do reconstruction using partial

shares. That is, if the interpolation threshold is 15, and the server collected shares from 20 decryptors, it will only use 15 of them and ignore the remaining 5 shares. Furthermore, as we only have a single set of decryptors, when the server collects shares from $U \subseteq \mathcal{D}$, it computes a single set of interpolation coefficients from U and uses it to do linear combinations on all the shares. This linear combination of all the shares can be done in parallel. In contrast, BBGLR requires the server to compute different sets of interpolation coefficients to reconstruct the pairwise masks (one set for each client).

Simulation framework. We integrade all of Flamingo's code into ABIDES [BHB20b, BHB20a], which is an open-source high-fidelity simulator designed for AI research in financial markets (e.g., stock exchanges). ABIDES is a great fit as it supports tens of thousands of clients interacting with a server to facilitate transactions (and in our case to compute sums). It also supports configurable pairwise network latencies.

2.2.10 Experimental evaluation

In this section we answer the following questions:

- What are Flamingo's concrete server and client costs, and how long does it take Flamingo to complete one and multiple iterations of aggregation?
- Can Flamingo train a realistic neural network?
- How does Flamingo compare to the state of the art in terms of the quality of the results and running time?

We implement the following baselines:

Non-private baseline. We implement a server that simply sums up the inputs it receives from clients. During a particular iteration, each of the clients sends a vector to the server. These vectors are in the clear, and may be any sort of value (e.g. floating points), unlike Flamingo, which requires data to be masked positive integers. The server tolerates dropouts, as Flamingo does, and aggregates only the vectors from clients who respond before the timeout.

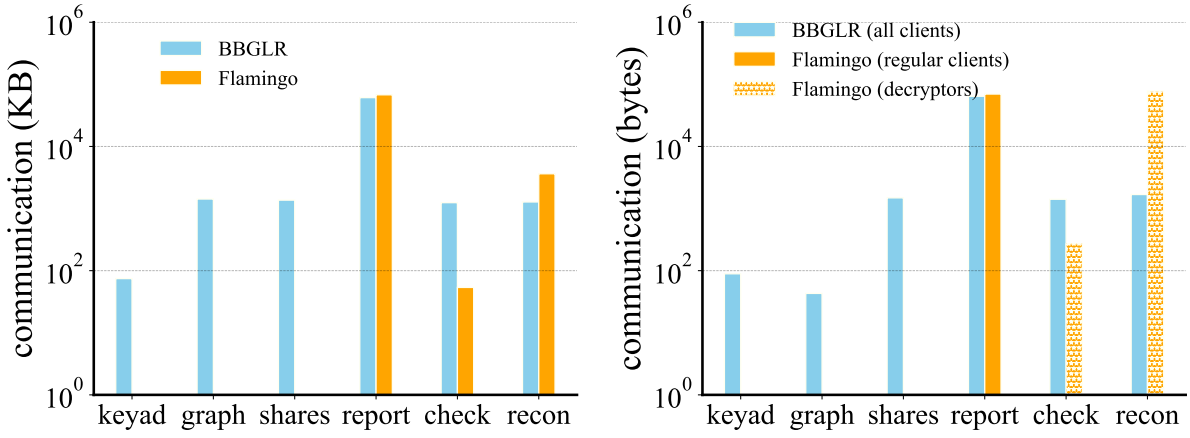
BBGLR. For BGGLR, we implement Algorithm 3 in their paper [BBG+20] with a slight variation that significantly improves BBGLR's running time, although that might introduce security issues (i.e., we are making this baseline's performance better than it is in reality, even if it means it is no longer secure). Our specific change is that we allow clients to drop out during the graph establishment procedure and simply construct a graph with the clients that respond in time. BBGLR (originally) requires that no client drops out during graph establishment to ensure that the graph is connected. Without this modification, BBGLR's server has to wait for all the clients to respond and is severely impacted by the long tail of the client response distribution—which makes our experiments take an unreasonable amount of time.

Experimental environment

Prior works focus their evaluation on the server's costs. While this is an important aspect (and we also evaluate it), a key contributor to the end-to-end completion time of the aggregation (and of the federated learning training) is the number of round trips between clients and the server. This is especially true for geodistributed clients.

To faithfully evaluate real network conditions, we run the ABIDES simulator [BHB20a] on a server with 40 Intel Xeon E5-2660 v3 (2.60GHz) CPUs and 200 GB DDR4 memory. Note that in practice, client devices are usually less powerful than the experiment machine. ABIDES supports the cubic network delay model [HRX08]: the latency consists of a base delay (a range), plus a jitter that controls the percentage of messages that arrive within a given time (i.e., the shape of the delay distribution tail). We set the base delay to the "global" setting in ABIDES's default parameters (the range is 21 microseconds to 53 milliseconds), and use the default parameters for the jitter.

Both Flamingo and BBGLR work in steps; each step is a complete round between server and clients. We define a *waiting* period for each step of the protocol. During the waiting period, the server receives messages from clients and puts the received messages in a message pool. When the waiting period is over, a *timeout* is triggered and the server processes the messages in the pool, and proceeds to the next step. The reason that we do not let the server send and receive messages at the same time is that, in some steps (in both BBGLR and Flamingo), the results sent to the clients



(a) Server communication

(b) Client communication

Figure 2.6: Communication costs for different steps in a single summation over 1K clients for Flamingo and BBGLR.

depend on all the received messages and cannot be processed in a streaming fashion. For example, the server must decide on the set of online clients before sending the request to reconstruct the shares.

Secure aggregation costs and completion time

This section provides microbenchmarks for summation tasks performed by BBGLR and Flamingo. Client inputs are 16K-dimensional vectors with 32-bit entries. For parameters, unless specified differently later, we set N to 10K and the number of decryptors to 60 in Flamingo; and set the number of neighbors to $4 \log n_t$ for both Flamingo and BBGLR (for BBGLR, this choice satisfies the constraints in Lemma 4.7 [BBG$^+$20]). In Figures 2.6 and 2.7, "keyad" is the step for exchanging keys, "graph" is the step for clients to send their choices of neighbors, "share" is the step for clients to shares their secrets to their neighbors (marked as 1–8 in their Algorithm 3). The steps "report", "check" and "recon" in Flamingo are described in Section 2.2.7; in BBGLR, these steps correspond to the last three round trips in a summation marked as 8–14 in their Algorithm 3.

Communication costs. Figure 2.6 gives the communication cost for a single summation. The total cost per aggregation for BBGLR and Flamingo are similar. This is because Flamingo's extra cost over BBGLR at the report step is roughly the message size that BBGLR has in their three-step

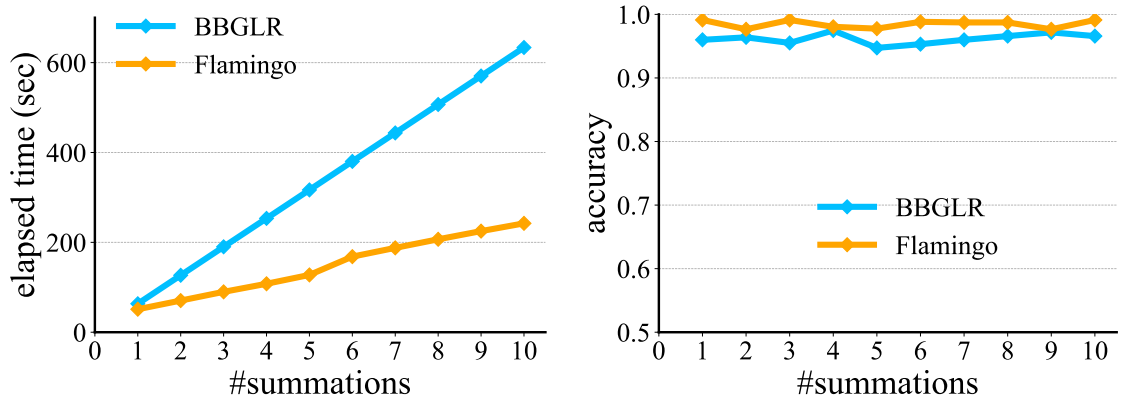
CPU costs	keyad	graph	share	report	check	recon
Server (sec)						
BBGLR	0.11	0.27	0.09	0.09	0.08	0.76
Flamingo	—	—	—	0.24	—	2.30
Client (sec)						
BBGLR	<0.01	<0.01	<0.01	0.02	0.01	<0.01
Flamingo						
Regular clients	—	—	—	0.22	—	—
Decryptors	—	—	—	—	0.10	0.56

Figure 2.7: Single-threaded microbenchmarks averaged over 10 runs for server and client computation for a single summation over 1K clients. "<" means less than.

setup; this is also reflected in the asymptotic cost analysis of Figure 2.5. In short, compared to BBGLR, Flamingo has fewer round trips with roughly the same total server communication. For clients, the story is more nuanced: each client has a slightly higher cost in Flamingo than in BBGLR during the report step, as clients in Flamingo need to append ciphertexts to the vector. However, in the reconstruction step, clients who are not decryptors will not need to send or receive any messages. Each decryptor incurs communication that is slightly larger than sending one input vector. Note that the vector size only affects the report step.

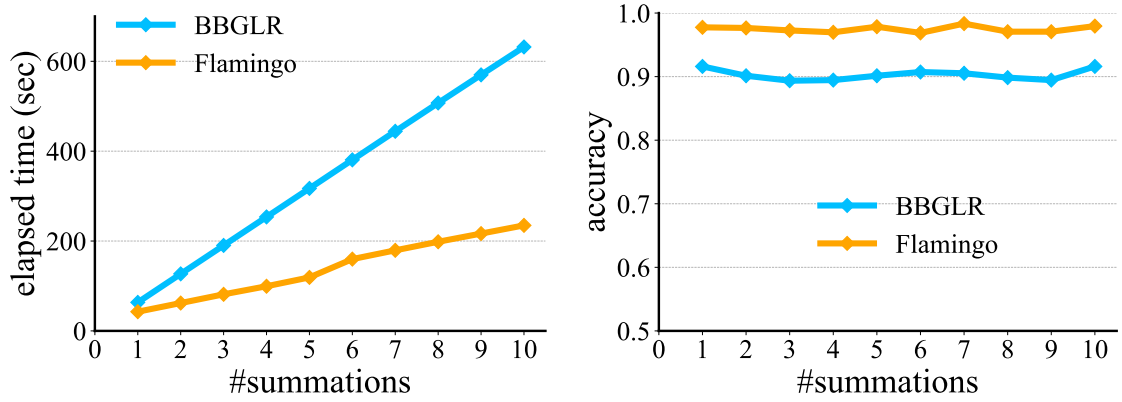
Computation costs. We first microbenchmark a single summation with 1K clients, and set δ to 1% (i.e., up to 1% of clients can drop out in any given step). This value of δ results in a server waiting time of 10 seconds. Figure 2.7 gives the results. The report step in Flamingo has slightly larger server and client costs than BBGLR because clients need to generate the graph "on-the-fly". In BBGLR, the graph is already established in the first three steps and stored for the report and reconstruction step. For server reconstruction time, Flamingo is slightly more costly than BBGLR because of the additional elliptic curve operations. The main takeaway is that while Flamingo's computational costs are slightly higher than BBGLR, these additional costs have negligible impact on completion time owing to the much larger effect of network delay, as we show next.

Aggregation completion time. To showcase how waiting time w affects dropouts (which consequently affects the sum accuracy), we use two waiting times, 5 seconds and 10 seconds. The runtime for an aggregation depends on the timeout for each step, the simulated network delay, and the server and



(a) Runtime with $w = 10$.

(b) Sum accuracy τ; $w = 10$.



(c) Runtime with $w = 5$.

(d) Sum accuracy τ; $w = 5$.

Figure 2.8: End-to-end completion time and accuracy of 10 secure aggregation rounds with 1K clients. The elapsed time is the finishing time of round t. For Flamingo, round 1 includes all of the costs of its one-time setup, and between round 5 and 6 Flamingo performs a secret key transfer.

client computation time. Figure 2.8a and 2.8c show the overall completion time across 10 iterations of aggregations. A shorter waiting time makes the training faster, but it also means that there are more dropouts, which in turn leads to more computation during reconstruction. As a result, the overall runtime for the two cases are similar. On 1K clients, Flamingo achieves a 3× improvement over BBGLR; for Flamingo's cost we included its one-time setup and one secret key transfer. If the key transfer is performed less frequently, the improvement will be more significant.

The cost of the DKG procedure (part of the setup and which we also added to the first iteration in Figure 2.8) is shown in Figure 2.9. A complete DKG takes less than 10 seconds as the number of

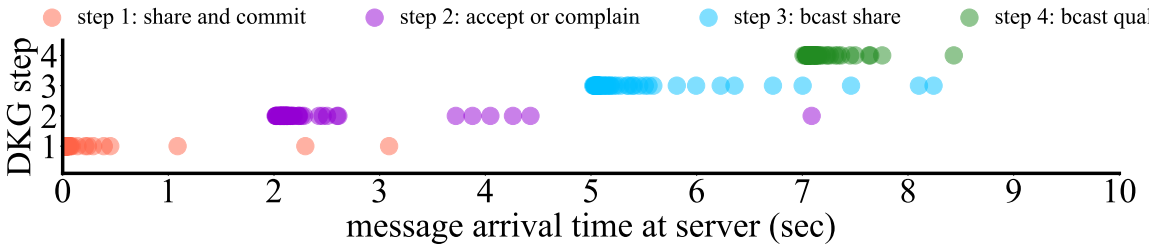


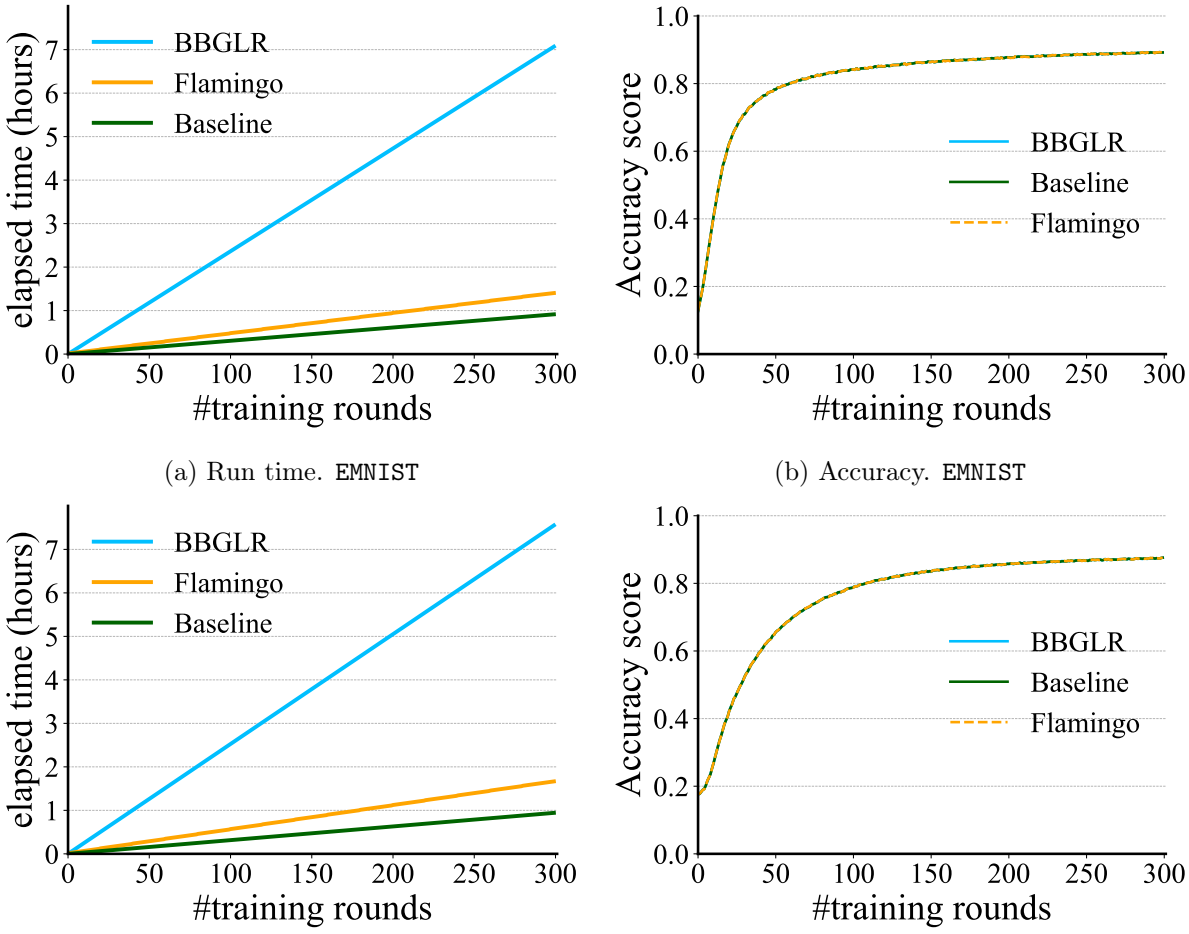
Figure 2.9: Generating shares of the secret key among 60 decryptors. The four steps are described in Section 2.3.4 and given as part (1) in Π_{DKG} in Appendix A.2.2.

decryptors is not large and we allow decryptor dropouts. For 60 decryptors, the local computation performed by each decryptor during the DKG is 2 seconds.

Summation accuracy. Figure 2.8b and 2.8d show that Flamingo achieves better sum accuracy τ (defined in §2.2.3) than BBGLR when they start a summation with the same number of clients. When the waiting time is shorter, as in Figure 2.8d, in each step there are more clients excluded from the summation and therefore the discrepancy between Flamingo and BBGLR grows larger.

Feasibility of a full private training session

We implement the federated learning algorithm `FedAvg` [MMR+17] on the non-private baseline. We also use this algorithm for Flamingo and BBGLR but replace its aggregation step with either Flamingo or BBGLR to produce a secure version. Inside of `FedAvg`, we use a multilayer perceptron for image classification. Computation of the weights is done separately by each client on local data, and then aggregated by the server to update a global model. The server then sends the global model back to the clients. The number of training iterations that clients perform on their local data is referred to as an *epoch*. We evaluated Flamingo on epochs of size 5, 10, and 20. We found that often, a larger epoch was correlated with faster convergence of `FedAvg` to some "maximum" accuracy score. Additionally, because our neural network model calculations run very fast—there was, on average, less than a second difference between clients' model fitting times for different epochs—and because Flamingo and the baselines were willing to wait for clients' inputs for up to 10 seconds, the epoch size did not affect their overall runtime.



(a) Run time. `EMNIST`

(b) Accuracy. `EMNIST`

(c) Run time. `CIFAR100`

(d) Accuracy. `CIFAR100`

Figure 2.10: Evaluation for full training sessions on `EMNIST` and `CIFAR100` datasets. The number of clients per round is 128, the batch and epoch sizes for `FedAvg` are 10 and 20, respectively. Flamingo's setup cost is included during the first round, and it performs a secret key transfer every 20 rounds, which adds to the total run time. The accuracy score is TensorFlow's sparse categorical accuracy score [AAB+15].

We use two of TensorFlow's federated datasets [AAB+15]: (1) `EMNIST`, the Extended MNIST letter dataset from the Leaf repository [CDW+18, CDW+]; and (2) `CIFAR100`, from the CIFAR-100 tiny images dataset [KNH, Kri09]. The `EMNIST` dataset has ∼340K training/∼40K test samples, each with a square of 28×28 pixels and 10 classes (digits). Each client has ∼226 samples. During training, we use weight vectors with 8K 32-bit entries. The `CIFAR100` dataset has 50K training/10K test samples, each with a square of 32×32 pixels and 100 classes. Each pixel additionally has red/blue/green values. Each client has 100 samples. To achieve good accuracy for the `CIFAR100` dataset, we use a more complex convolutional neural network than we do for the `EMNIST` dataset, with extra layers to build the model, normalize inputs between layers, and handle activation functions and overfitting. This results in longer weight vectors, with 500K 32-bit entries.

We randomly divide the datasets equally among 128 clients to create local data. Local models are trained with small batch sizes. In Flamingo and BBGLR, all weights (often floating point numbers) are encoded as positive integers. We do this by adding a large positive constant, multiplying by 2^{12}, and truncating the weight to an unsigned 32-bit integer. Figure 2.10 shows the result with $\delta = 1\%$.

Running time. From Figures 2.10b and 2.10d, we see that the `EMNIST` and `CIFAR100` datasets do not converge until about iteration 150 and 200, respectively, though their accuracy continues to improve slightly after that. Figures 2.10a and 2.10c show Flamingo's running time is about 5.5× lower (i.e., better) than BBGLR for `EMNIST` and 4.8× for `CIFAR100` and about 1.4× higher (i.e., worse) than the non-private baseline for `EMNIST` and 1.7× for `CIFAR100`. We believe these results provide evidence that Flamingo is an effective secure aggregation protocol for multi-iteration settings such as those required in federated learning.

Training accuracy. We measure training accuracy with TensorFlow's sparse categorical accuracy score, which is derived based on the resulting model's performance on test data. Due to the way in which we encode floating points as integers, a small amount of precision from the weights is lost each iteration. We compare the accuracy of Flamingo and BBGLR's final global model to a model trained on the same datasets with the baseline version of `FedAvg` (which works over floating points) in Figures 2.10b and 2.10d. We find that the encoding itself does not measurably affect accuracy.

2.2.11 Related work

In this section we discuss alternative approaches to compute private sums and the reasons why they do not fit well in the setting of federated learning. Readers may also be interested in a recent survey of this area [MOJC23].

Pairwise masking. Bonawitz et al. [BIK+17] and follow up works [BBG+20, SGA21], of which BBGLR [BBG+20] is the state-of-the-art, adopt the idea of DC networks [Cha88] in which pairwise masks are used to hide individuals' inputs. Such a construction is critical for both client-side and server-side efficiency: first, since the vectors are long, one-time pad is the most efficient way to encrypt a vector; second, the server just needs to add up the vectors, which achieves optimal server computation (even without privacy, the server at least has to do a similar sum). Furthermore, pairwise masking protocols support flexible input vectors, i.e., one can choose any b (the number of bits for each component in the vector) as desired. Flamingo improves on this line of work by reducing the overall round trip complexity for multiple sums.

MPC. Works like FastSecAgg [KRKR20] use a secret-sharing based MPC to compute sums, which tolerates dropouts, but it has high communication as the inputs in federated learning are large. Other results use non-interactive MPC protocols for addition [SHY+22, SGA21] where all the clients establish shares of zero during the setup. And then when the vectors of clients are requested, each client uses the share to mask the vector and sends it to the server. However, to mask long vectors, the clients need to establish many shares of zeros, which is communication-expensive. Such shares cannot be reused over multiple summations (which is precisely what we address with Flamingo). Furthermore, the non-interactive protocols are not resilient against even one dropout client.

Additively homomorphic encryption. One can construct a single-server aggregation protocol using threshold additive homomorphic encryption [EDG14, MDC16, PBB09, PBBL11, TBA+19, DSM22]. Each client encrypts its vector as a ciphertext under the public key of the threshold scheme, and sends it to the committee. The committee adds the ciphertexts from all the clients and gives the result to the server. However, this does not work well for large inputs (like the large weight vectors

found in federated learning) because encrypting the vector (say, using Paillier or a lattice-based scheme) and performing the threshold decryption will be very expensive.

A recent work [SSV+22] uses LWE-based homomorphic PRGs. This is an elegant approach but it has higher computation and communication costs than works based on pairwise masking, including Flamingo. The higher cost stems from one having to choose parameters (e.g., vector length and the size of each component) that satisfy the LWE assumption, and particularly the large LWE modulus that is required.

Multi-iteration setting. Recent work [GPS+22] designs a new multi-iteration secure aggregation protocol with reusable secrets that is very different from Flamingo's design. The protocol works well for small input domains (e.g., vectors with small values) but cannot efficiently handle large domains as it requires brute forcing a discrete log during decryption. In contrast, Flamingo does not have any restriction on the input domain. A variant of the above work can also accommodate arbitrary input domains by relying on DDH-based class groups [CL15] (a more involved assumption than DDH).

2.3 Armadillo: Secure aggregation with disruption resistance

2.3.1 Background and summary of contributions

In this section, we focus on how to prevent malicious clients from disrupting the server's aggregation result. Federated learning has a large number of training participants, and it is highly likely that some clients are compromised. However, even a single malicious client could distort the overall aggregation result.

While disruption resistance has been studied in the context of secure aggregation, most of existing aggregation systems that tolerates disruption work under a strictly weaker trust model than the setting we consider in this dissertation. They require two or more non-colluding servers with at least one being trusted to achieve their goals [CGB17, AGJ+22], yet, in real-world federated learning deployment the single-server architecture seems to be the only choice: an organization that runs the training tasks either internally operate their own servers while ensuring isolation among the servers, which is rarely realistic; or they set up external servers elsewhere, which introduces significant engineering overhead and operational risks. Indeed, industry precedent has consistently relied on the single-server model [BIK+17, BBG+20]. The few existing single-server aggregation protocols that tolerate disruption unfortunately have high costs: Eiffel [CGJvdM22] has a per-client computational workload quadratic in the number of clients n, and ACORN-robust [BGL+22] despite its modest client cost has too many rounds to be efficient—a single aggregation takes logarithmic in n and concretely 10–20 rounds. Some other works [LBV+23] settle for a relatively weak guarantee where the aggregation has to abort once disruption is detected. That is, there was essentially no affordable disruptive-resistant secure aggregation schemes under practical adversarial models.

This gap drives our work: we show that ensuring privacy for clients while resisting disruption can be achieved in only 3 rounds, even in the presence of an adversary controlling the server and a subset of clients (up to some threshold). Our system Armadillo guarantees the following properties: 1) privacy, i.e., the server learns at most the sum of inputs from clients but nothing else, 2) robustness, i.e., the server, if following our protocol, is ensured to get the sum regardless how clients passively drop out or actively disrupt the aggregation. (A malicious client can also use invalid input

to disrupt the result, but we show that our system can seamlessly integerate with existing efficient input validation techniques, resulting in a complete disruption-resistant system.) Our reduction in round complexity comes at only a slight increase in client computational time, and when integrated with input validation techniques, client computation in Armadillo is on par with that in ACORN-robust. Across a range of client population sizes and adversarial fractions, Armadillo outperforms ACORN-robust by 3× to 7× in rounds.

To achieve disruption resistance, Armadillo uses a generic paradigm: take a secure aggregation protocol, every client sends to the server a proof that every step of their execution has been done correctly and the server verifies the proof. The core challenge is to make this efficient because cryptographic proofs are expensive; in fact, this is more challenging than it may seem—most of the prior works [BGL+22, LBV+23, CGJvdM22] making strides towards robustness do not follow this paradigm. We have two key design ideas that make clients and the server computation lightweight. First, we design a secure aggregation protocol in which the bulk of computation is simple linear computation, and importantly, it is sufficient to get robustness as long as the clients prove the correctness of the linear part (which is computationally efficient). Then, we structure all these proof statements (together with the input validation) as a single inner-product relation, so that with existing proof systems [BBDBM18] the server can batch verify n proofs at a cost logarithmic in n.

Armadillo additionally has other beneficial properties in federated learning setting: stateless participation of clients, many aggregations followed by a one-time setup, and easy handling for dropouts. We discuss them in detail in Section 2.3.4.

2.3.2 Setting, threat model, and goals

The problem setting and communication model are the same as Flamingo. We have a star-topology network with a single server coordinating all the communication, and we assume a PKI that stores public keys of all the clients.

For the threat model, same as in Flamingo, we assume for each aggregation, at most η fraction of

n clients may be malicious, and up to δ fraction may passively drop out (in addition to malicious dropouts). A major difference is that the server here is assumed to be honest-but-curious (i.e., it follows the protocol but may try to learn more than what it is supposed to learn). This threat model captures those settings where the server wants to learn the aggregation results but concerns about disruptive clients who cause disproportional damage to the result. Many existing aggregation systems assume one or more semi-honest servers as well [CGB17, AGJ+22, BGL+22].

The system goals include what has been achieved in Flamingo (dropout resilience and privacy), with a slight difference being that privacy is against an honest-but-curious server. As Flamingo, Armadillo supports multiple iterations of aggregation; but different from Flamingo, Armadillo does not require any clients to hold states across iterations.

Definition 2.3.1 (Privacy)**.** We say an aggregation protocol has privacy against a semi-honest server if the protocol realizes the ideal functionality in Figure 2.11.

The salient aspect here is achieving disruption resistance: the coalition of malicious clients can affect the aggregation result only by misreporting their private inputs. This is formalized in Definition 2.3.2 below.

Definition 2.3.2 (Disruption resistance [CGB17])**.** Let f be an aggregation function that takes in n inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Let $\mathcal{Z}$ be a user-defined set of valid inputs, i.e., we say $\mathbf{x}$ is a valid input for f if $\mathbf{x} \in \mathcal{Z}$. We say that an n-client aggregation protocol has *disruption resistance* if, when the server follows the protocol, for every number of m clients (with $0 \leq m \leq n$) and for every choice of honest client's inputs $\mathcal{I}_{honest} \in \mathcal{Z}^{n-m}$, the server outputs a value in the set $\{f(\mathcal{I}_{honest}, \mathcal{I}_{malicious}) \mid \mathcal{I}_{malicious} \in \mathcal{Z}^{n-m}\}$. If we relax $\mathcal{Z}$ to be the set of all possible inputs for f, then we say the protocol has *robustness*.

2.3.3 Cryptographic building blocks

Notation. Let $[z]$ denote the set $\{1, 2, \ldots, z\}$. We use $[a, b]$ to denote the set $\{x \in \mathbb{N} : a \leq x \leq b\}$. We use bold lowercase letters (e.g. $\mathbf{u}$) to denote vectors and bold upper case letters (e.g., $\mathbf{A}$) to denote matrices. Unless specified, vectors are column vectors. Given a value α and a vector $\mathbf{v}$,

Functionality $\mathcal{F}$

Parties: A set of n clients $P_1, \ldots, P_n$ and a server S.
Notation: Let corruption rate be η and dropout rate be δ, both among $\mathcal{P} = \{P_1, \ldots, P_n\}$. Let $\mathcal{A}$ be the adversary corrupting S and the set of clients of size ηn, Cor.

- $\mathcal{F}$ and $\mathcal{A}$ receive a set of dropout clients $\mathcal{O} \subset \mathcal{P}$ where $|\mathcal{O}|/|\mathcal{P}| \leq \delta$. $\mathcal{F}$ receives $\mathbf{x}_i$ of client $P_i \in \mathcal{P} \backslash \mathcal{O}$.
- $\mathcal{F}$ asks $\mathcal{A}$ for a set $\mathcal{E}$ with the requirements that: $|\mathcal{E} \cup \mathcal{O}|/n \leq \delta$.
- If $\mathcal{A}$ replies $\mathcal{F}$ with a set $\mathcal{E}$ that satisfies the requirement, then $\mathcal{F}$ outputs $\mathbf{z} = \sum_{i \in \mathcal{P} \backslash (\mathcal{E} \cup \mathcal{O} \cup \mathsf{Cor})} \mathbf{x}_i$ to $\mathcal{A}$. Otherwise, $\mathcal{F}$ send `terminate` to all parties.

Figure 2.11: Ideal functionality for one aggregation. We follow the definition in prior works [BIK+17, BBG+20] assuming an oracle gives a dropout set to $\mathcal{F}$ and adversary $\mathcal{A}$ can also query the oracle.

we use $\alpha \mathbf{v}$ to denote multiplying α to every coordinate of $\mathbf{v}$. For distribution $\mathcal{D}$, we use $a \leftarrow \mathcal{D}$ to denote sampling a from $\mathcal{D}$. For a vector $\mathbf{v}$, we use $\lfloor \mathbf{v} \rceil_c$ to denote rounding each entry of $\mathbf{v}$ to nearest multiples of c. For two vectors $\mathbf{v}_1$ of length ℓ_1, $\mathbf{v}_2$ of length ℓ_2, we use $\mathbf{v}_1 | \mathbf{v}_2$ to denote the concatenation of them which is a vector of length $\ell_1 + \ell_2$. We use $\|\mathbf{v}\|_2$ to denote L_2 norm of $\mathbf{v}$ and use $\|\mathbf{v}\|_\infty$ to denote the largest entry in $\mathbf{v}$. We use $\mathbb{F}$ to denote a field.

Regev's encryption. Our construction utilizes the key-and-message homomorphism Regev's encryption [Reg05]; we give the details below. The Regev's scheme is parameterized by a security parameter λ, a plaintext modulus p, and a ciphertext modulus q, and number of LWE samples m. Given a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\lambda$, the encryption of a vector $\mathbf{x} \in \mathbb{Z}_p^m$ is

$$(\mathbf{A}, \mathbf{c}) := (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}),$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times \lambda}$ is a random matrix ($m > \lambda$), $\mathbf{e} \xleftarrow{\$} \chi^m$ is an error vector and χ is a discrete Gaussian distribution, and $\Delta := \lfloor q/p \rfloor$. Decryption is computed as $(\mathbf{c} - \mathbf{A}\mathbf{s}) \bmod q$ and rounding each entry to the nearest multiples of Δ, and then divide the rounding result by Δ. The decrypted result is correct if entries in $\mathbf{e}$ are less than $\Delta/2$.

Packed secret sharing. In standard Shamir secret sharing [Sha79], a secret $\rho \in \mathbb{F}$ is hidden as the constant term of a polynomial $p(x) = a_0 + a_1 x + \cdots + a_t x^d$ where $a_0 = \rho$ and $a_1, \ldots, a_d$ are randomly sampled from $\mathbb{F}$. Given n parties, the share for party $i \in [n]$ is $p(i)$, and any subset of at

least $d+1$ parties can reconstruct ρ and any subset of d shares are independently random.

In packed secret sharing [FY92], one can hide multiple secrets using a single polynomial. Specifically, let $\mathbb{F}$ be a field of size at least $2n$ and k be the number of secrets packed in one sharing. Packed Shamir secret sharing of $(v_1, \ldots, v_k) \in \mathbb{F}^k$ first chooses a random polynomial $p(\cdot) \in \mathbb{F}[X]$ of degree at most $d+k-1$ subject to $p(0) = v_1, \ldots, p(-k+1) = v_k$, and then sets the share ρ_i for party i to be $\rho_i = p(i)$ for all $i \in [n]$. Reconstruction of a degree-$(d+k-1)$ sharing requires at least $d+k$ shares from $\rho_1, \ldots, \rho_n$. Note that the corruption threshold is now d even if the degree is $d+k-1$, i.e., any d shares are independently random, but any $d+1$ shares are not.

Shamir sharing testing. Looking ahead, we will also use a probabilistic test for Shamir's secret shares, called SCRAPE test [CD17]. To check if $(\rho_1, \ldots, \rho_n) \in \mathbb{F}^n$ is a Shamir sharing over $\mathbb{F}$ of degree d (namely there exists a polynomial p of degree $\leq d$ such that $p(i) = \rho_i$ for $i = 1, \ldots, n$), one can sample $w_1, \ldots, w_n$ uniformly from the dual code to the Reed-Solomon code and check if $w_1\rho_1 + \cdots + w_n\rho_n = 0$ in $\mathbb{F}$.

To elaborate, let $c_i := \prod_{j \in [n] \setminus \{i\}} (i-j)^{-1}$ and $m^*(X) := \sum_{i=0}^{n-d-2} m_i \cdot X^i \leftarrow_{\$} \mathbb{F}[X]_{\leq n-d-2}$ (a random polynomial over $\mathbb{F}$ of degree at most $n-d-2$). Now, let $\mathbf{w} := (c_1 \cdot m^*(1), \ldots, c_n \cdot m^*(n))$ and $\boldsymbol{\rho} := (\rho_1, \ldots, \rho_n)$. Then,

- If there exists $p \in \mathbb{F}[X]_{\leq d}$ such that $\rho_i = p(i)$ for all $i \in [n]$, then $\langle \mathbf{w}, \boldsymbol{\rho} \rangle = 0$.
- Otherwise, $\Pr[\langle \mathbf{w}, \boldsymbol{\rho} \rangle = 0] = 1/|\mathbb{F}|$.

In other words, if $(\rho_1, \ldots, \rho_n)$ is not a Shamir sharing of degree d then the test will only pass with probability $1/|\mathbb{F}|$.

Pedersen and vector commitment. Let $\mathbb{G}$ be a group of order q, and G, H be two generators in $\mathbb{G}$. A Pedersen commitment to a value $v \in \mathbb{Z}_q$ is computed as $\mathsf{com}_G(v) := G^v H^r$, where the commitment randomness r is uniformly chosen from $\mathbb{Z}_q$. We use $\mathsf{com}_G(\cdot)$ notation because later in our protocol we compute commitments with different generators.

We can also commit to a vector $\mathbf{v} = (v_1, \ldots, v_L) \in \mathbb{Z}_q^L$ as follows: let $\mathbf{G} = (G_1, \ldots, G_L)$ be a list of

L random generators in $\mathbb{G}$, define $\mathsf{com}_{\mathbf{G}}(\mathbf{v}) := G_1^{v_1} \cdots G_L^{v_L} \cdot H^r$, where r is randomly chosen from $\mathbb{Z}_q$; our notation $\mathsf{com}_{\mathbf{G}}(\cdot)$ implictly assumes a public H and a private r are included. In a special case that we will get to in Section 2.3.4 and 2.3.4, we do not include randomness in the commitment.

Inner-product proof. The inner product argument is an efficient proof system for the following relation: given two vector commitments $\mathsf{com}(\mathbf{a}), \mathsf{com}(\mathbf{b})$ known to both prover and verifier and a public value c, the prover can convince the verifier that $\langle \mathbf{a}, \mathbf{b} \rangle = c$. Bulletproof [BBDBM18] gives non-interactive inner-product proof system with proof size $O(\log L)$ and prover/verifier cost $O(L)$.

For ease of presentation later, we introduce the following notations for proof. A proof system Π consists of a tuple of algorithms $(\mathcal{P}, \mathcal{V})$ run between a prover and verifier. An argument to prove can be described with public inputs/outputs io, a statement to be proved st, and a private witness wt. Given a proof system Π, the prover can generate a proof $\pi \leftarrow \Pi.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier checks the proof by $b \leftarrow \Pi.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi)$ where $b \in \{0, 1\}$ indicates rejecting or accepting π. For example, for proving inner product of $\mathbf{a}$ and $\mathbf{b}$, we set the constraint system to be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{a}), \mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : (\mathbf{a}, \mathbf{b})\}.$$

Denote the inner product proof system as Π_{ip}, the prover runs $\pi \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier runs $b \leftarrow \Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$. The algorithms $\Pi_{\text{ip}}.\mathcal{P}$ and $\Pi_{\text{ip}}.\mathcal{V}$ both have complexity linear to the length of $\mathbf{a}$ (or $\mathbf{b}$) and π has logarithmic length of $\mathbf{a}$ (or $\mathbf{b}$). We will also prove *linear-relation*, and we denote the proof system as Π_{linear} and the constraint system will be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : \mathbf{b}\}.$$

To differentiate the two proof systems, we call the former (that needs to commit to both vectors in the inner product) as quadratic proof and the later (that only needs to commit to one vector in the inner product) as linear proof.

Below we briefly review two small primitives we will use later in our protocol.

Public key encryption. A public key encryption (PKE) scheme allows a sender to encrypt a message using a recipient's public key such that only the corresponding secret key can decrypt it. Formally, a PKE scheme consists of three algorithms:

- $\mathrm{Gen}(1^\lambda) \to (pk, sk)$: on input a security parameter λ, generates a public-secret key pair;
- $\mathrm{Enc}(pk, m) \to c$: on input a message m and public key pk, output a ciphertext c;
- $\mathrm{Dec}(sk, c) \to m$: on input a ciphertext c and secret key sk, output a message m.

We require correctness (i.e., decryption recovers the original message) and semantic security (i.e., ciphertexts leak no information about the plaintext without the secret key).

MAC. A message authentication code (MAC) is a symmetric primitive used to ensure data integrity and authenticity. It consists of two algorithms:

- $\mathsf{Mac}(k, m) \to t$: on input a key k and a message m, outputs a tag t;
- $\mathsf{MacVer}(k, m, t) \to b$: on input a key k and message m and a tag t, outputs $b \in \{0, 1\}$ indicates whether t is an invalid tag or valid for m under key k.

MACs are required to be unforgeable: no efficient adversary can produce a valid tag for a new message without knowledge of the secret key k.

2.3.4 Protocol design

Now we describe our construction for computing a single sum (one iteration in the training). Our full protocol is shown in Figures B.1 and B.2; below we describe our main technical ideas. We discuss computing multiple sums and the related security consideration in Section 2.2.8.

A two-layer secure aggregation

The key idea is to reduce an aggregation for long vectors to an aggregation for short vectors. To substantiate this idea, we utilize the key-and-message homomorphism of Regev's encryption.

Given two Regev ciphertexts $(\mathbf{A}, \mathbf{c}_1), (\mathbf{A}, \mathbf{c}_2)$ of vectors $\mathbf{x}_1, \mathbf{x}_2$ under the key $\mathbf{s}_1, \mathbf{s}_2$ with noise $\mathbf{e}_1, \mathbf{e}_2$,

the tuple $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ is an encryption of $\mathbf{x}_1 + \mathbf{x}_2$ under the key $\mathbf{s}_1 + \mathbf{s}_2$. The ciphertext $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ can be properly decrypted if $\mathbf{e}_1 + \mathbf{e}_2$ is small. Note that computing $\mathbf{c}_1 + \mathbf{c}_2$ is very efficient—it is simply vector addition.

We define a tuple of algorithms $(\mathrm{Enc}, \mathrm{Dec})$ parameterized by $(p, q, \lambda, m, \mathbf{A} \in \mathbb{Z}_q^{m \times \lambda})$ as follows:

- $\mathrm{Enc}(\mathbf{s}, \mathbf{x}) \to \mathbf{y}$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a message $\mathbf{x} \in \mathbb{Z}_p^m$, output $\mathbf{y} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}$, where $\Delta = \lfloor q/p \rfloor$.
- $\mathrm{Dec}(\mathbf{s}, \mathbf{y}) \to \mathbf{x}'$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a ciphertext $\mathbf{y} \in \mathbb{Z}_q^m$, output $\mathbf{x}' := \lfloor \mathbf{y} - \mathbf{A}\mathbf{s} \rceil_\Delta$.

Now we decribe how our protocol work at a high level. Each client $i \in [n]$, holding an input vector $\mathbf{x}_i \in \mathbb{Z}_p^\ell$ (we can set $m = \ell$ in Enc/Dec), samples a Regev encryption key $\mathbf{s}_i \in \mathbb{Z}_q^\lambda$ and sends the encrypted vector $\mathbf{y}_i = \mathrm{Enc}(\mathbf{s}_i, \mathbf{x}_i) := \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i$ to the server. Note that $\lambda \ll \ell$. The server computes the sum of $\mathbf{y}_i$'s as

$$\mathbf{y} := \sum_{i \in [n]} \mathbf{y}_i = \sum_{i \in [n]} \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i = \mathbf{A} \sum_{i \in [n]} \mathbf{s}_i + \Delta \sum_{i \in [n]} \mathbf{x}_i + \sum_{i \in [n]} \mathbf{e}_i,$$

To reconstruct $\sum_{i \in [n]} \mathbf{x}_i$, the server needs $\mathbf{s} := \sum_{i=1}^n \mathbf{s}_i$ to decrypt $\mathbf{y}$, and the decryption succeeds if $\sum_{i \in [n]} \mathbf{e}_i < \Delta/2$. We call the sum of Regev's ciphertexts $\mathbf{y}_i$'s as *outer aggregation*, and next we discuss *inner aggregation* where the server gets sum of $\mathbf{s}_i$'s.

The inner aggregation could be instantiated with a naive secure multi-party computation over the n clients: each client i secret shares $\mathbf{s}_i$ coordinate-wise to all the other clients (the shares are encrypted using public keys of the recipient clients and sent through the server), and each client adds up the shares which is then sent to the server for reconstruction of $\mathbf{s}$. However, this naive approach has per client communication $O(n\lambda)$ and server communication $O(n^2\lambda)$. The former is too much for a client given λ is a security parameter and is typically from a few hundreds to a thousand; and the latter is too much for the server because it is quadratic in n. We reduce the communication complexity with two techniques: 1) let $C \ll n$, sample C clients from the whole population as *decryptors* to assist

with the secure computation[4]; 2) each client i uses packed secret sharing (§2.3.3) to share its secret vector $\mathbf{s}_i$ to the decryptors. The combination of these two techniques reduces the communication complexity per client to $O(\lambda)$ and per decryptor to $O(n)$, and the server communication complexity is $O(Cn)$.

This inner-outer aggregation has a key advantage in handling dropouts, unlike the pairwise masking approach used in prior works [BIK+17, BBG+20, MWA+23], which incurs extra rounds. Specifically, if a client drops out while sending $\mathbf{y}_i$ or shares in the outer aggregation, the server can safely ignore the client without affecting subsequent steps. If a decryptor client drops out during the inner aggregation, the server can still reconstruct $\mathbf{s}$ due to the threshold nature of secret sharing.

In the next few sections, we discuss how to make this simple protocol robust against malicious clients (including the decryptors) in two parts: 1) proof of linear computation and 2) an agreement protocol for inner aggregation.

Remark 4. As observed in a few works in orthogonal areas [HHCG+23, DPC23], Regev's encryption remains secure even if $\mathbf{A}$ is made public and the same matrix $\mathbf{A}$ is used to encrypt polynomially many messages, as long as the secret key $\mathbf{s}$ and the noise $\mathbf{e}$ are independently chosen in each instance of encryption. Later in our protocol, $\mathbf{A}$ is a public random matrix and it can be generated by a trusted setup [DKIR21, clo] (who generates a seed and the parties use PRG to expand the seed to matrix $\mathbf{A}$). Since $\mathbf{A}$ can be reused, so this setup only needs to run once.

Proof of client computation

Our high-level idea is "commit-and-proof": each client sends to the server commitments to its private values (e.g., commitment to $\mathbf{s}_i$) together with a proof of the following relations. Let $\mathbf{F}, \mathbf{G}, \mathbf{H}$ be vectors of group generators in $\mathbb{G}$ of length λ, ℓ, ℓ respectively. Suppose client i sends to the server $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i), \mathsf{com}_{\mathbf{G}}(\mathbf{e}_i), \mathsf{com}_{\mathbf{H}}(\mathbf{x}_i)$, in addition to $\mathbf{y}_i$ as specified in the outer aggregation. The client proves to the server that:

1. For the outer aggregation, $\mathbf{y}_i := \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i \bmod q$, with $\mathbf{e}_i$ having small L_∞ norm.

[4]We show that C can be polylogarithmic in n to have this work (§2.2.8).

2. For the inner aggregation, the client secret-shares $\mathbf{s}_i$ to the decryptors using a polynomial of degree d (the degree d is fixed by the threat model parameters).

Next, we express these requirements (except the norm condition which is non-linear) as inner-product relations. We will address proving the norms in Section 2.3.4. We set LWE modulus q to match the field size of the commit-and-proof system.

Proving the first statement. At the first glance, we need to prove that each coordinate of $\mathbf{y}_i$ equals the corresponding coordinate of the RHS computation result. This would require ℓ proofs, one for each coordinate. We instead use a polynomial checking technique from Schwartz-Zippel Lemma to compress the proofs to a single one. In particular, if we want to check if two vectors of length ℓ over $\mathbb{Z}_q$ are equal, we view each vector (e.g., $\mathbf{y}_i$) as coefficients of a degree-ℓ polynomial and check if the evaluation of the two polynomials on a random point are equal. If they are indeed not equal, then the evaluation will be different except probability ℓ/q. Formally, let $r \in \mathbb{Z}_q$ be a random challenge value picked by the server (who is the verifier), and let $\mathbf{r} = (r^0, r^1, \ldots, r^{\ell-1})$. Let $c = \langle \mathbf{y}_i, \mathbf{r} \rangle$, and c is a public value since $\mathbf{y}_i$ and $\mathbf{r}$ are both public (known to both the client and the server). If the client can prove to the server that

$$c = \langle \mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta\mathbf{r},\ \mathbf{s}_i \mid \mathbf{e}_i \mid \mathbf{x}_i \rangle \text{ in } \mathbb{Z}_q,$$

then the server will be convinced that $\mathbf{y}_i := \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i$, and there will only be ℓ/q probability that the client is dishonest but the server is convinced. The inner product argument comes from the following:

$$\begin{aligned}
\langle \mathbf{y}_i, \mathbf{r} \rangle =& \langle \mathbf{A}\mathbf{s}_i, \mathbf{r} \rangle + \langle \mathbf{e}_i, \mathbf{r} \rangle + \Delta \langle \mathbf{x}_i, \mathbf{r} \rangle \\
=& \langle \mathbf{A}^\top \mathbf{r}, \mathbf{s}_i \rangle + \langle \mathbf{r}, \mathbf{e}_i \rangle + \langle \Delta\mathbf{r}, \mathbf{x}_i \rangle \\
=& \langle \mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta\mathbf{r},\ \mathbf{s}_i \mid \mathbf{e}_i \mid \mathbf{x}_i \rangle.
\end{aligned}$$

Also, note that $\mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta\mathbf{r}$ is public, the client only needs to do a linear proof, where the witness is under the commitment $\mathsf{com}_{\mathbf{F}|\mathbf{G}|\mathbf{H}}(\mathbf{s}_i|\mathbf{e}_i|\mathbf{x}_i)$.

Proving the second statement. Recall that the client sends $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i)$ to the server in the outer ag-

gregation, and now we want to ensure that the shares that the decryptors received (for the inner aggregation) are indeed the Shamir shares of this committed $\mathbf{s}_i$. Here we can exactly use the SCRAPE test (§2.3.3) to express this constraint as an inner product relation; this test seamlessly works with packed secret sharing.

Formally, supppose client i has a packed Shamir sharing of $\mathbf{s}_i$ as a vector of length C (recall that there are C decryptors)

$$\boldsymbol{\rho}_i = (\rho_i^{(1)}, \ldots, \rho_i^{(C)}),$$

which the client claims is a sharing of degree d over $\mathbb{Z}_q$. We observe that checking if $\boldsymbol{\rho}_i$ is a packed sharing of $\mathbf{s}_i$ is equivalent to checking if $(\boldsymbol{\rho}_i \mid \mathbf{s}_i)$ is a sharing of length $C+\lambda$ of a degree-d polynomial. Therefore, we let the client commit to $\boldsymbol{\rho}_i$ under a new generator vector $\mathbf{K} = (K_1, \ldots, K_C) \in \mathbb{G}^C$, and sends $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ to the server in the outer aggregation as well. Then the client invokes a linear-relation proof that

$$\langle \boldsymbol{\rho}_i \mid \mathbf{s}_i,\ \mathbf{w} \rangle = 0 \text{ in } \mathbb{Z}_q,$$

where $\mathbf{w} := (w^{(1)}, \ldots, w^{(C+\lambda)})$ is sampled uniformly random from some code space (details in §2.3.3) and is public (known to both the server and client). In our setting, we cannot let the client choose $\mathbf{w}$ since the client may be malicious, so we apply the Fiat-Shamir transform and have client i derive $\mathbf{w}$ by hashing $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) \cdot \mathsf{com}_{\mathbf{F}}(\mathbf{s}_i)$.

Up to this point, we have not guaranteed the shares received by the decryptors are consistent with the commitment $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$. The reason is that the client could in fact send to a decryptor a share (under the encryption) that is different from what was committed to. Therefore, instead of having the client send $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ to the server, we have the client send commitments to each coordinate of $\boldsymbol{\rho}_i$, namely $\mathsf{com}_{K_1}(\rho_i^{(1)}), \ldots, \mathsf{com}_{K_C}(\rho_i^{(C)})$. Since the shares are random and in a sufficiently large space, we will compute the commitment to a share ρ simply as K_j^{ρ}. The server can still verify the proof for packed sharing, as it can compute the vector commitment $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ from the individual C commitments as

$$\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) = \mathsf{com}_{K_1}(\rho_i^{(1)}) \cdots \mathsf{com}_{K_C}(\rho_i^{(C)}).$$

For those clients whose proofs are valid (for both Enc computation and SCRAPE test), the server forwards their commitments to the corresponding decryptors where $\rho_i^{(j)}$ is intended for the j-th decryptor. Then each decryptor j verifies if the received share (after decryption) is consistent with the commitment $\mathsf{com}_{K_j}(\rho_i^{(j)})$.

Agreement protocol for decryptors

So far, each decryptor can identify a set of *valid* shares by verifiying the commitments. However, this alone is not enough for the server to obtain the correct sum, as illustrated in the following example. Suppose there are three clients P_1, P_2, and P_3, with P_3 being malicious, and three decryptors H_1, H_2, and H_3, with H_3 being malicious. Each client shares a secret using 2-out-of-3 Shamir sharing. If P_3 sends a valid share to H_1 but an invalid share to H_2, the decryptors will form the sets of clients with valid shares as follows:

- H_1 forms $\{P_1, P_2, P_3\}$,
- H_2 forms $\{P_1, P_2\}$,
- H_3 can form any arbitrary set, e.g., $\{P_3\}$.

If each decryptor adds up the valid shares locally, the resulting values will not reconstruct the sum of the secrets from any set of clients. The server can only reconstruct a meaningful sum if the decryptors adds up shares from the *same set* of clients.

Our goal is to ensure that all honest decryptors agree on the same set of clients with valid shares. Although this seems like a consensus problem that may require many rounds, we leverage the observation that the communication through a semi-honest server is equivalent to a broadcast channel between the decryptors. We borrow ideas of proof of decryption from Benhamouda et al. [BHK+24] to build an agreement protocol to ensure decryptors agree on the same set of clients.

The clients in the first round send their shares to decryptors encrypted under the decryptors' public keys. If a decryptor j decrypts a share ρ and finds that it is not consistent with the commitment K_j^ρ, then it sends a *verifiable complaint* to the server: the share ρ (in the clear) and a zero-knowledge

proof of decryption relative to pk_j (which is published at PKI). The server can verify the complaint (because it knows the share in the clear and the ciphertext from the first round) and informs all the decryptors of the lying clients. Every decryptor removes the lying clients from their set. Since the server is semi-honest, all the decryptors at this point should agree on the same set. Finally, they can add up the shares of this set and send the result to the server. The server can reconstruct the sum, applying error correction if there are bogus shares from malicious decryptors.

Since the public-key encryption is black box in this agreement protocol, we can instantiate it with any scheme as long as it has fast proof of decryption. See Appendix B.2 for details.

Integrating with proof of norms

Our protocol can be seamlessly integrated with existing vector norm validation techniques [BGL+22, GHL22]. Now we describe how a client proves a vector $\mathbf{x}_i$ has bounded L_2, L_∞ norms; this also applies to proving norms of the LWE error vector $\mathbf{e}_i$ that we mentioned earlier. Below, we represent an integer as a number in $\mathbb{Z}_q$ where the negative numbers are in $(-q/2, -1]$ and the positive numbers are in $[0, q/2]$. We will use the term "ZKPoK" to denote zero-knowledge proof of knowledge.

We first introduce a primitive that we will use as blackbox called *approximate proof of L_∞-smallness* [LNS21]. This proof system has a large multiplicative gap γ between the L_∞ norm of the vector and what the prover can prove: for a vector $\mathbf{a}$ and a bound B that we wish to impose on $\mathbf{a}$, either the prover is honest and $\|\mathbf{a}\|_\infty < B$, or the prover is dishonest and $\|\mathbf{a}\|_\infty < \gamma B$. We defer the construction of approximate proof to Appendix B.1, but in short, an approximate proof for length-ℓ vector just invokes a linear proof of length $\ell + \sigma$ where σ is a security parameter usually taken as 256.

The idea underlying proof of L_2 and L_∞ norms is the same: Assuming the bound B we want to prove is much smaller than q, to prove a number x is bounded by B in a field $\mathbb{Z}_q$, we find four

integers a_1, a_2, a_3, a_4 such that[5]

$$B^2 = x^2 + a_1^2 + a_2^2 + a_3^2 + a_4^2,$$

and the proof consists of two parts:

- Use the underlying commit-and-prove systems to show that this equality holds modulo q,
- Use approximate proof to show that the numbers $x, a_1, \ldots, a_4$ are small enough so that they do not trigger a wraparound modulo q.

For wraparound, we require $x, a_1, \ldots, a_4$ are all smaller than $\sqrt{q/10}$ (this is a necessary condition for $x^2 + a_1^2 + \cdots + a_4^2 < q/2$ to hold).

To prove a length-ℓ vector $\mathbf{x}$ has L_2 norm bounded by B, we find four integers $a_1, \ldots, a_4$ such that

$$B^2 = \|\mathbf{x}\|_2^2 + a_1^2 + \cdots + a_4^2,$$

and define $\mathbf{v} = \mathbf{x}|(a_1, \ldots, a_4)$. The prover provides a ZKPoK that $\|\mathbf{v}\|_2^2 = B^2 \bmod q$, and provides a ZKPoK showing that $\|\mathbf{v}\|_\infty < \sqrt{q/2(\ell+4)}$. The former can be done using an inner-product proof system, and the latter can be done using approximate proof.

Proving L_∞ is similar but with a difference that we need to find four squares for each entry of the vector $\mathbf{b} - \mathbf{x}$ where $\mathbf{b} := B \cdot \mathbf{1}$. Namely, there exist $\mathbf{a}_1, \ldots, \mathbf{a}_4$ such that

$$\mathbf{x} + \mathbf{a}_1 \circ \mathbf{a}_1 + \cdots + \mathbf{a}_4 \circ \mathbf{a}_4 = \mathbf{b}.$$

We apply Schwartz-Zippel to this equation. Let $\mathbf{r} := (r^0, r^1, \ldots, r^{\ell-1})$ where r is random in $\mathbb{Z}_q$,

$$\langle \mathbf{r}, \mathbf{x} + \mathbf{a}_1 \circ \mathbf{a}_1 + \cdots + \mathbf{a}_4 \circ \mathbf{a}_4 \rangle = \langle \mathbf{r}, \mathbf{b} \rangle.$$

[5] Any positive integer can be decomposed into four squares and finding the four squares can be done in $O(\log^2 a)$ time [RS86].

We rewrite the above equation as

$$\langle \mathbf{r} \circ \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{r} \circ \mathbf{a}_1, \mathbf{a}_1 \rangle + \langle \mathbf{r} \circ \mathbf{a}_2, \mathbf{a}_2 \rangle + \langle \mathbf{r} \circ \mathbf{a}_3, \mathbf{a}_3 \rangle + \langle \mathbf{r} \circ \mathbf{a}_4, \mathbf{a}_4 \rangle = \langle \mathbf{r}, \mathbf{b} \rangle, \tag{2.1}$$

which is exactly an inner product relation:

$$\langle \mathbf{r} \circ \mathbf{x} | \mathbf{r} \circ \mathbf{a}_1 | \cdots | \mathbf{r} \circ \mathbf{a}_4, \mathbf{x} | \mathbf{a}_1 | \cdots | \mathbf{a}_4 \rangle = \langle \mathbf{r}, \mathbf{b} \rangle. \tag{2.2}$$

Note that the RHS of Equation 2.2 is a public value. We commit to $\mathbf{z} = \mathbf{x}|\mathbf{a}_1|\cdots|\mathbf{a}_4$, and commit to $\mathbf{r}' \circ \mathbf{z}$ where $\mathbf{r}' = \mathbf{r}|\mathbf{r}|\mathbf{r}|\mathbf{r}|\mathbf{r}$, and proving the relation in Equation 2.2 goes in two steps: first, invoke any inner-product proof system for Equation 2.2, and second, prove that the messages underlying the two commitments indeed have a linear relation w.r.t. $\mathbf{r}'$. For the latter, suppose the prover commits to $\mathbf{y}$ claimed to be $\mathbf{r}' \circ \mathbf{z}$. The prover can prove $\mathbf{y} = \mathbf{r}' \circ \mathbf{z}$ using an inner product proof as follows. Let α be a random challenge in $\mathbb{Z}_q$, and $\boldsymbol{\alpha} = (1, \alpha, \ldots, \alpha^{\ell-1})$. Then

$$\langle \boldsymbol{\alpha}, \mathbf{y} \rangle = \langle \boldsymbol{\alpha}, \mathbf{r}' \circ \mathbf{z} \rangle = \langle \boldsymbol{\alpha} \circ \mathbf{r}', \mathbf{z} \rangle,$$

and we rewrite as

$$\langle \boldsymbol{\alpha} - \boldsymbol{\alpha} \circ \mathbf{r}', \mathbf{y} - \mathbf{z} \rangle = 0. \tag{2.3}$$

Note that $\boldsymbol{\alpha} \circ \mathbf{r}'$ is public, and the commitment to $\mathbf{y} - \mathbf{z}$ can be easily computed from the individual commitments to $\mathbf{y}$ and $\mathbf{z}$. Now we can invoke a linear proof for Equation 2.3, with the public vector being $\boldsymbol{\alpha} - \boldsymbol{\alpha} \circ \mathbf{r}'$ and the secret vector being $\mathbf{y} - \mathbf{z}$.

To summarize, proving L_2 norm requires a length-$(\ell + 4)$ quadratic proof, and a length-$(\ell + 4)$ approximate proof which can be instantiated using a length-$(\ell + 260)$ linear proof. Proving L_∞ norm requires a length-(5ℓ) quadratic proof, and a length-(5ℓ) linear proof. We can further reduce the proof cost by decomposing to three numbers if one can afford one bit leakage [GHL22], and therefore the vector length for proof of L_2 norm becomes $\ell + 3$ and the vector length for proof of L_∞ norm becomes 4ℓ.

Combined with the linear proof of encryption and secret sharing, the overall cost for proof is summarized as follows.

Lemma 2.3.3 (Cost for proof of encryption and norms)**.** *Given a set of parameters* (λ, ℓ, q, C) *and let* $\mathbf{s} \in \mathbb{Z}_q^\lambda, \mathbf{s} \in \mathbb{Z}_q^C, \mathbf{M} \in \mathbb{Z}_q^{\lambda \times C}, \mathbf{x} \in \mathbb{Z}_q^\ell$*. Let* $\mathbb{G}$ *be a group of size* q*. Let* Δ *be a constant. Let*

$$\begin{aligned}\mathbb{CS}_{enc} : \{&\mathsf{io} : (\mathsf{com}(\mathbf{s}), \mathsf{com}(\mathbf{x}), \mathsf{com}(\mathbf{e})),\\ &\mathsf{st} : \mathbf{y} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}, \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2),\\ &\|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{e}\|_\infty < B_{\mathbf{e}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{s}, \mathbf{x}, \mathbf{e})\}.\end{aligned}$$

There exist a commit-and-proof protocol Π_{enc} *(Appendix B.3) with group* $\mathbb{G}$ *of order* q *for proving the above statement with the following cost, dominated by the inner-product proof (either linear proof or quadratic proof) invocations:*

- *Quadratic proofs: 2 length-*(4ℓ)*, 1 length-*ℓ*,*
- *Linear proofs: 2 length-*(4ℓ)*, 1 length-*ℓ*, 1 length-*$(2\ell + \lambda)$*, 1 length-*$(\lambda + C)$*,*

where we omit the lower order terms and write e.g., $\ell + 256$ *as* ℓ*.*

Electing decryptors

The focus of previous sections is the aggregation protocol and we therefore assumed there already exists a set $\mathcal{D}$ of decryptors with corruption rate $\eta_{\mathcal{D}}$ before the aggregation starts. In fact, $\eta_{\mathcal{D}}$ depends on the exogenous parameter η^* and how we sample $\mathcal{D}$. In this section, we describe the sampling and the relation between $\eta_{\mathcal{D}}$ and η^*. We will leave the analysis of how large the set $\mathcal{D}$ need to be in Section 2.2.8.

We use a sampling protocol by Alon et al. [ANOS24], and this does not require additional assumptions like random beacon used in a prior work Flamingo [MWA+23]. Their core building block is Feige's election protocol [Fei99]: suppose for now we have a public bulletin board, and we want to sample (roughly) X out of N clients. We initialize N/X bins on the bulletin board. Each client

jumps into a bin independently at random (malicious clients may not do it randomly). Then we take the smallest bin as the set of decryptors. This simple sampling actually ensures that when the total population has a corruption rate η, the sampled set (with size at most X) also has a corruption rate bounded by $\frac{\eta^*}{1-2\epsilon}$, except probability $N \cdot e^{\Omega(-\epsilon^4 X)}$ [Fei99, KMSW22]. The protocol of Alon et al. [ANOS24] eliminates the need for the bulletin board and extends Feige to work exactly under our communication model (§2.2.3); their protocol is more sophisticated but the key take away is that: if we have corruption rate η^* over all clients, then their protocol will elect a set with corruption rate $\eta_{\mathcal{D}} \leq 2\eta^*$ except with negligible probability, as long as the target X is at least polylogarithmic of N.

2.3.5 Security analysis and parameter selection

Parameters. The system Armadillo has a set of parameters listed below. First, n is the number of clients per round. (λ, ℓ, p, q) are LWE parameters (for the outer aggregation), (C, d, λ) are secret-sharing parameters (for the inner aggregation), where C is the number of shares (which equals to the number of decryptors), d is the degree of the secret-sharing polynomial and λ is the number of secrets. $B_{\mathbf{X}}(L_\infty), B_{\mathbf{X}}(L_2), B_{\mathbf{e}}(L_\infty)$ are bounds on norms. The parameters n and the norm bounds depends on the machine learning setting which is orthogonal to security analysis. For (λ, ℓ, p, q), we can choose any secure instance of LWE [APS15, CCSL24, DSDGR20].

The choice of C and d is shown below. Recall that in our protocol (§2.3.4), each client secret-shares a vector of length λ using a polynomial of degree d. We must have

$$
\begin{aligned}
&d - \lambda > C \cdot \eta_{\mathcal{D}} \quad \text{by security of packed secret sharing,} \\
&d < C(1 - \delta_{\mathcal{D}}) \quad \text{necessary condition to reconstruct the secret.} \\
&C \cdot \eta_{\mathcal{D}} < \frac{(1 - \delta_{\mathcal{D}})C - d + 1}{2} \quad \text{in order to do error correction.}
\end{aligned}
$$

We combine the equations and get

$$C\eta_{\mathcal{D}} + \lambda < d < C(1 - \delta_{\mathcal{D}} - 2\eta_{\mathcal{D}}). \tag{2.4}$$

We need to ensure C is at least polylogarithmic in the total population so that $\eta_{\mathcal{D}} \leq 2\eta$ (§2.3.4), and we choose $C > \lambda/(1 - \delta_{\mathcal{D}} - 3\eta_{\mathcal{D}})$ and set d accordingly.

Let Φ denote the protocol in Figures B.1 and B.2 i.e., Φ is a protocol running between n clients and the server, where each client i has inputs $\mathbf{x}_i \in \mathbb{Z}_q^\ell$ and the server has no input. We describe the ideal functionality of Φ in Figure 2.11.

Theorem 2.3.4. *Let $(\delta, \eta, \delta_{\mathcal{D}}, \eta_{\mathcal{D}})$ be threat model parameters defined in Section 2.2.3, and $\mathcal{D}$ is a set of clients randomly subsampled from total N clients prior to aggregation. Let (λ, ℓ, p, q) be LWE parameters. If (λ, ℓ, p, q) is a secure LWE instance, and $|\mathcal{D}| > \lambda/(1 - \delta_{\mathcal{D}} - 3\eta_{\mathcal{D}})$, then under the communication model defined in Section 2.2.3, protocol Φ (Fig.B.1,B.2) securely realizes ideal functionality $\mathcal{F}_{sum}$ (Fig.2.11) in the presence of a semi-honest adversary controlling the server, η fraction of n clients for each aggregation and $\eta_{\mathcal{D}}$ fraction of clients in $\mathcal{D}$. Also, Π satisfies disruption resistance in Definition 2.3.2.*

2.3.6 Implementation and optimization

Sparse LWE. The bulk of server-side decryption lies in computing the matrix-vector product $\mathbf{A} \cdot \mathbf{s}$. When $\mathbf{A}$ is sparse, that is, most of its entries are zero, this computation can be significantly accelerated. To realize this, we can replace the standard LWE assumption used in Regev's encryption scheme with Sparse LWE assumption; by increasing the length of the secret to $1.5\times$ of its size in the LWE instance, we maintain the same security level with the LWE instance [JLS24].

Multi-exponentiation. Naively computing commitments to a length-m vector requires $m + 1$ group exponentiations and m group multiplications (§2.3.3). We can reduce the number of group exponentiations to sublinear in m using the Pippenger algorithm, given in Lemma 2.3.5. In short, the Pippenger algorithm requires only a small number of expensive group exponentiation (e.g., σ is typically no larger than 256) by increasing the cheap group multiplication by a factor of σ. Therefore, we can use Pippenger for Pedersen vector commitment in any of our inner-product proofs.

Lemma 2.3.5 (Complexity of Pippenger algorithm [Pip, Boo17, Fen23])**.** *Let $\mathbb{G}$ be a group of order $q \approx 2^\sigma$, and $G_1, \ldots, G_m$ be m generators of $\mathbb{G}$. Given $v_1, \ldots, v_m \in \mathbb{Z}_q$, Pippenger algorithm can*

compute $G_1^{v_1} \cdots G_n^{v_m}$ *using* $\frac{2\sigma m}{\log m}$ *group multiplications and* σ *group exponentiations.*

Optimistic batch verification. There are two verification steps we can optimize. First, each helper verifies that each online client i's share ρ_i matches the commitment $\mathsf{com}_K(\rho_i) = K^{\rho_i}$ where K is the generator of $\mathbb{G}$. Doing this individually for each share requires up to n group exponentiations; exponentiation is an expensive operation. We can apply batch verification technique [BGR98] to reduce the number of exponentiations to sublinear in n. Say the helper has shares $\rho_1, \ldots, \rho_n$ and it wants to batch verify if they match with $\mathsf{com}_K(\rho_1), \ldots, \mathsf{com}_K(\rho_n)$ respectively. It samples random values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{Z}_q$ where q is group order and checks if $\alpha_1\rho_1 + \cdots + \alpha_n\rho_n$ matches $\mathsf{com}_K(\rho_1)^{\alpha_1} \cdots \mathsf{com}_K(\rho_n)^{\alpha_n}$. The batch verification only needs a length-n inner product on $\mathbb{Z}_q$ and a single length-n multi-exponentiation.

The batching technique can also be applied to proof verification at the server. For this we open the black box of the inner-product proof system and utilize a property of Bulletproof [BBDBM18] that allows much faster batch verfication than verifying each proof individually. To verify a quadratic proof of length ℓ, the verifier computation can be abstracted as[6]

$$\mathbf{G}^{\mathbf{z}} \stackrel{?}{=} \mathbf{P}^{\mathbf{r}}, \tag{2.5}$$

where $\mathbf{G}$ is a vector of group generators of length-2ℓ and $\mathbf{P}$ is the proof transcript which consists of $2\log\ell$ group elements. The exponents $\mathbf{z}, \mathbf{r}$ only depend on the verifier challenges. Verifying n quadratic proofs naively would require the server to compute n multi-exponentiation of length 2ℓ (LHS of 2.5), and n multi-exponentiation of length $2\log\ell$ (RHS of 2.5).

The key idea is that the LHS of 2.5 can be computed for a batch of n proofs using a single multi-exponentiation instead of n multi-exponentiations; the insight is that all the proofs share the same generators $\mathbf{G}$. To batch verify n proofs with exponents $\mathbf{z}_1, \ldots, \mathbf{z}_n$, the verifier (server) can sample

[6]The linear proof can be abstracted the same way with length-ℓ vector $\mathbf{G}$.

random $\alpha_1, \ldots, \alpha_n$ and computes $\mathbf{z}' = \alpha_1 \mathbf{z}_1 + \cdots + \alpha_n \mathbf{z}_n$ and checks if

$$\mathbf{G}^{\mathbf{z}'} \stackrel{?}{=} \mathbf{P}_1^{\alpha_1 \mathbf{r}_1} \cdots \mathbf{P}_n^{\alpha_n \mathbf{r}_n}, \tag{2.6}$$

where $\mathbf{P}_i$ and $\mathbf{r}_i$ are from the RHS of equation 2.5 for client i's proof. For LHS of 2.6, the server computes a single multi-exponentiation of length 2ℓ. For RHS of 2.6, the server computes a single multi-exponentiation of length $2n \log \ell$.

In both the above cases, if there exists malicious clients such that the batch verification returns false, then the helper (or the server) can divide the shares (or the proofs) into smaller groups and recursively apply batch verification until it identifies the malicious client.

Other details. For machine learning applications, it is not necessary to find four squares that sum exactly to the input. Instead, it suffices to find four squares whose sum approximates the input closely. Therefore, we can use greedy algorithm to find the squares. For every integer in $[0, 2^{16}]$, the greedy approach consistently finds four squares whose sum is within 10 of the original value.

2.3.7 Evaluation

In this section, we provide benchmarks to answer the following questions:

- What are Armadillo's concrete costs of the clients and the server, for aggregation and proofs, respectively?
- What is the cost of the decryptors and how does it compare to the cost of regular clients?
- How does Armadillo's performance compare to prior robust secure aggregation protocols?

Selecting a proper baseline. The most relevant work is ACORN-robust [BGL+22]: they provide the same input validation but achieve robustness in a very different way. ACORN-robust follows the pairwise masking approach in Bell et al. [BBG+20], but they additionally have a dispute protocol to iteratively find cheating clients and remove their inputs from the aggregation result. The other prior work with disruption resistance is Eiffel [CGJvdM22], but their per-client work is $O(n^2\ell)$

which is not feasible for computationally restricted devices. Other works like RoFL [LBV+23], ACORN-detect [BGL+22] have strictly weaker property: the server has to abort the aggregation once a malicious client has been detected. Therefore, we identify ACORN-robust as the only valid baseline.

Libraries and testbed. We implement our protocol using Rust. We choose Rust because many zero-knowledge proof systems are implemented using this language, due to its performance, safety, and ecosystem support. This makes it easy to swap in alternative proof systems if needed. We instantiate the inner-product proof using Bulletproof [BBDBM18]. We use the dalek library [dVYA18] for elliptic curve cryptography. We run our experiments on a 2.4GHz Apple M2 CPU.

Concrete parameter selection. The proof system implemented in dalek library is based on Ristretto group. So we set the LWE modulus q equal to the group order which is a 253-bit prime.[7] To control the noise growth in ciphertext computation, we require $n \cdot B_{\mathbf{e}} < \Delta/2$, i.e., $2pnB_{\mathbf{e}} < q$. According to the LWE estimator [APS15], we can set λ to be 256, $\mathbf{s}$ uniform in $\mathbb{Z}_q$, and the error uniform in $\mathbb{Z}_{2^{2^{14}}}$, which gives 132 bits of security; this supports plaintext modulus $p = 2^{16}$ with summation up to 5K clients, sufficient for federated learning applications [KMA+21, Table 2].

Following prior work [BGL+22], we use input lengths close to powers of two for benchmarking. Since the inner-product proof system is most efficient when applied to vectors of length exactly equal to a power of two, we set the aggregation input to be slightly shorter than a power of two, and this ensures that the input to the proof system has length exactly power-of-two, minimizing the overhead from padding.

Central theme in evaluation. Experiments in the following sections will substantiate a central argument we make: while Armadillo has similar computational cost as ACORN-robust, the reduction in round complexity plays a crucial role in lowering the end-to-end runtime. The key to this improvement lies in the interplay between the allowed dropout rate and server waiting time, which we explain in detail in Section 2.3.7.

[7] One could use a LWE modulus that is much smaller than the group order of the proof system, but this requires additional L_∞ proofs and non-trivial changes for the Schwartz-Zippel compressing technique.

Input length ℓ approx.	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
Packed sharing in Armadillo (ms)	29.67	29.67	29.67	29.67	29.67
Feldman in ACORN (ms)	90.32	90.32	90.32	90.32	90.32
Masking in Armadillo (ms)	0.26	0.49	1.13	2.01	4.07
Masking in ACORN (ms)	1.69	3.81	9.41	15.53	31.45
Commitment (ms)	16.98	18.28	21.35	27.18	39.02
- commit to $\mathbf{s}$	1.31	1.31	1.31	1.31	1.31
- commit to $\mathbf{e}$	1.99	2.93	5.25	9.72	18.53
- commit to $\mathbf{x}$	1.19	1.55	2.30	3.66	6.69
- commit to shares in Armadillo	12.49	12.49	12.49	12.49	12.49
- commit to masks in ACORN	164.80	288.85	535.58	1038.54	2042.97
Proofs (sec)					
- L_∞ norm (§2.3.4)	1.74	2.40	4.78	7.04	14.12
- L_2 norm (§2.3.4)	0.09	0.17	0.33	0.64	1.26
- enc linear (§2.3.4)	0.08	0.16	0.56	0.61	1.22
- scrape test (§2.3.4)	0.80	0.80	0.80	0.80	0.80

Figure 2.12: Computation cost per client in Armadillo and ACORN-robust [BGL+22] varying input vector lengths.

Computation and communication

Regular client cost. We present the breakdown of a regular client's computational cost in Figure 2.12, varying the input length. The cost for decryptors is discussed separately later. For Armadillo, we set the number of decryptors as 512, which is more than needed in most cases (§2.2.3): with $\lambda = 256$, this can tolerate $\delta_{\mathcal{D}} + 3\eta_{\mathcal{D}} < 1/2$ (§2.3.5).

Since our baseline ACORN-robust shares similar types of computation with Armadillo, we present the costs for two systems jointly, dividing the process into four steps: input-independent secret sharing, input masking, commitment generation, and proof generation.[8] In each phase, we indicate whether ACORN-robust and Armadillo perform the same or different operations: shared operations are shown in black text, while differences are highlighted using distinct colors. Since the authors of ACORN-robust did not implement this protocol, we estimate their client's costs by extracting the types of operations and counting the number of operations for each type (e.g., how many

[8]In ACORN-robust, each client establishes input-independent pairwise secrets with $O(\log n)$ neighbors, expands them to $O(\log n)$ masks of input length, and performs Feldman secret sharing of the secrets. Microbenchmarks for ACORN-robust assume 40 neighbors per client (from [BBG+20]).

scalar multiplication on elliptic curve), and simulating them with Rust (the same language used for implementing ours). The constructions proving the of L_2 and L_∞ norms are largely identical across both protocols.

For both ACORN-robust and Armadillo, the bulk of the computation time is spent on proof generation; the other phases take only milliseconds. Armadillo has a 1–2 second additional work for the scrape test proof and the proof of encryption (§2.3.4), but as we show later (§2.3.7), this is a trade-off that yields significant gains: while clients spend slightly more time, our protocol requires fewer than one-third the number of rounds compared to theirs. Since round complexity is substantially important for run time in a setting where clients may drop out arbitrarily (evidenced in §2.3.7), this reduction leads to much overall performance gains that is far outweigh the effect of increased client computation.

A final complication is that ACORN-robust has an additional cheater identification phase to remove the effects of malicious clients from the aggregation result. Since this phase involves only sending small stored messages to the server (Algorithm 4 in [BGL+22]) and there is no cryptographic computation at clients, we do not depict them in Figure 2.12.

Decryptor cost. Per-decryptor cost is independent of the input length ℓ, and is linear to the number of clients n. Each decryptor will receive n ciphertexts and n commitments, decrypts each ciphertext and batch verify if the decryption results are consistent with the commitments.

We instantiate the public key encryption using RSA since it supports extremely simple proof of decryption (§B.2): the decryptor reveals the decryption result to the server, and the server checks the result using this decryptor's public key at PKI.

For RSA cryptosystem, each ciphertext decryption takes around 1.5 ms. For 1K clients, this is only 1.5 seconds per decryptor cost. Also, the decryptor only needs to send proof of decryption to the server for at most ηn clients.

Server cost. Armadillo's server computation depends on both input length ℓ and the number of

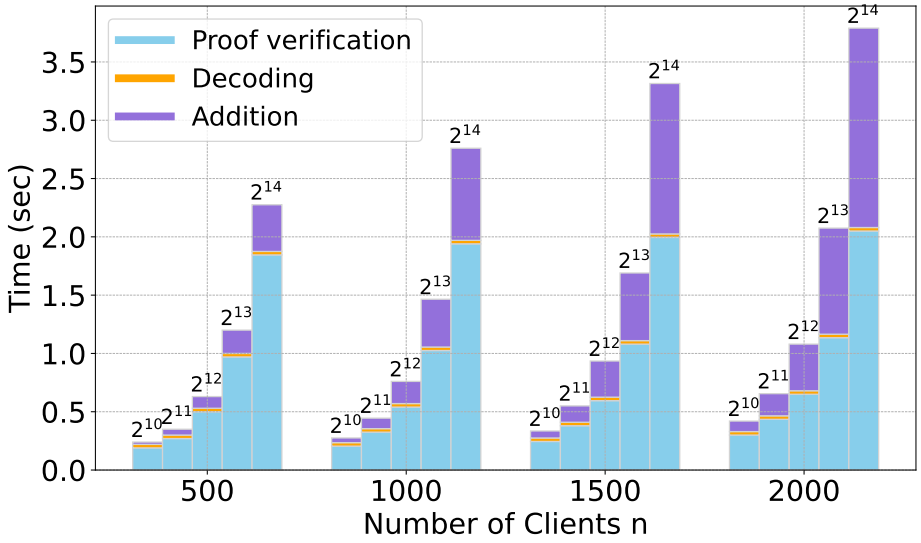


Figure 2.13: Server computation in Armadillo for different number of clients (indicated via x-axis) and different lengths of inputs (indicated on the top of bars).

clients n. Figure 2.13 breaks down the computation into aggregation (summing masked vectors and decoding using $\mathbf{s}$) and proof verification. As the number of clients increases, the proportion of time spent on proof verification decreases due to batching optimizations.

For ACORN-robust, the server performs the same proof verification for input norms and the vector additions as in Armadillo, and it additionally verifies Feldman commitments for every pairwise seed. We do not report their cost for Feldman commitment verification but it could become a bottleneck: for 1K clients with 40 neighbors each[9], the server performs $40^2 \cdot 1000$ group exponentiations.

Simulating communication rounds

Why rounds matter. When executing an interactive protocol in the server-client setting, it is a common strategy to fix the server's waiting time per round and drop any clients who are late. Typically, once a client is late, the client no longer participates in the rest of the rounds (because it may have lost the state necessary for future rounds).

This interaction pattern exacerbates the impact of round complexity: say a protocol is theoretically

[9]The number of neighbors is to guarantee privacy with respect to the dropout rate and malicious rate, and 40 is for 5% dropouts and 5% malicious clients which are the smallest rates considered in Bell et al. [BGL+22].

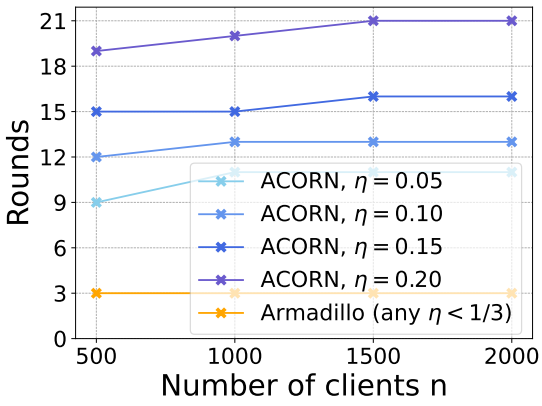


Figure 2.14: Number of rounds in Armadillo and ACORN-robust, varying malicious rate η.

designed to tolerate 10% dropouts over all rounds of an aggregation, if this protocol has many rounds, e.g., 20 rounds, then even 1% dropout on average per round can accumulate to 20% dropouts in total in which case the protocol just fails, i.e., the server does not get any aggregation result.

Comparison on rounds. Figure 2.14 shows the concrete number of rounds for ACORN-robust under different n and η based on their probabilistic analysis [BGL+22, Theorem 4.1]. For a 0.9 success probability, ACORN-robust requires up to 21 rounds (7× of Armadillo) in the worst setting ($n = 2000$, $\eta = 0.2$), compared to 3 rounds for Armadillo. In the best setting when $n = 500$ and $\eta = 0.05$, ACORN-robust still has 9 rounds (3× of Armadillo). Also, ACORN communicates with all clients in every round, while Armadillo involves all clients in the first round and only decryptors in the remaining two.

2.3.8 Related work

Multi-server setting. Disruption resistance is first considered in the context of aggregation statistics and most relevant works focus on the multi-server setting. Their high level idea is achieving privacy by having clients secret-share their inputs to the multiple servers, and the servers communicate to validate the inputs. Relevant systems include [CGB17, RSWP23, ZZW24, NLT24].

Our model is a star-topology model where a single server coordinates all the communication (the same setting as [BIK+17, BGL+22, MWA+23]), which differs fundamentally from the multi-server model. In the multi-server setting, powerful servers can handle heavy computation, such as the

$n\ell$ work for n clients and length-ℓ inputs under the secret-sharing solutions. For our star-topology communication model, computational and communication overhead must be minimized for the resource-constrained clients. In other words, the bottomline for client computation and communication is $O(n + \ell)$, and for example any protocol with $n\ell$ work at any client is not a practical solution under the star-topology model.

Single-server setting. Although there are many single-server secure aggregation protocols [BIK+17, BBG+20, SHY+22, SSV+22, MWA+23, GPS+22, LLPT23, KP24, BCGL+24], most of them do not provide disruption resistance. A few prior works [LBV+23, BGL+22] achieves a weaker notion of disruption resistance: disruption can be detected, but the server cannot reconstruct the sum result if there is an disruption. ACORN-robust [BGL+22] has only $\mathsf{polylog}(n)$ work per client, but requires a probabilistic cheater identification mechanism that requires $O(\log n)$ rounds. A recent work by Alon et al. [ANOS24] shows that any function can be securely computed with $\mathsf{polylog}(n)$ work per client and $\mathsf{poly}(n)$ work at the server, with $\mathsf{polylog}(n)$ rounds, assuming fully homomorphic encryption exists.

Our protocol (and [BGL+22, MWA+23]) works by each client sending its masked input of length ℓ to the server; then in the rest of the steps, the clients do computation *independent* of ℓ to decryptor the server unmask the sum. To compare, ACORN-robust has each client communicate with $\mathsf{polylog}(n)$ other clients where each of them doing $\mathsf{polylog}(n)$ work; Armadillo has each client communicate to a set of $\mathsf{polylog}(n)$ clients where the latter does linear work in n. In practice, n is smaller than input size ℓ so that the input size will dominate the client cost.

2.4 Comparison, discussion and limitations

Figure 2.15 provides a comprehensive comparison for the guarantees and efficiency of the two systems Flamingo and Armadillo with prior secure aggregation protocols. Below we discuss some extensions and limitations.

Computing aggregate statistics. We have focused on computing sums, but we can also compute other functions such as max/min using affine aggregatable encodings [BGI+14, CGB17, HIKR23].

	Client comm.	Client comp.	Server comm.	Server comp.	Rounds	Robustness	Input Val.
Effiel [CGJvdM22]	ℓn^2	ℓn^2	ℓn^3	ℓn^3	4	✓	Generic
RoFL [LBV+23]	$\ell + \log n$	$\ell \log n$	$\ell n + n \log n$	ℓn	6	×	L_2, L_∞
ACORN-detect [BGL+22]	$\ell + \log n$	$\ell \log n$	$\ell n + n \log n$	ℓn	7	×	L_2, L_∞
ACORN-robust [BGL+22]	$\ell + \log^2 n$	$\ell \log n + \log^2 n$	$\ell n + n \log^2 n$	$\ell n + n \log^2 n$	$6 + \log n$	✓	L_2, L_∞
Flamingo	Regular: $\ell + C$ Decryptor: $n + C$	Regular: $\ell + C$ Decryptor: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	×	N/A
Armadillo	Regular: $\ell + C$ Decryptor: $n + C$	Regular: $\ell + C$ Decryptor: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	✓	L_2, L_∞

Figure 2.15: Asymptotic communication and computation cost for one training iteration, where vector length is ℓ and number of clients per iteration is n; for simplicity, we omit the asymptotic notation $O(\cdot)$ in the table. In practice we have $n < \ell$ (§2.2.3). The round complexity excludes any setup that is one-time. We choose the baseline protocols that have similar properties as ours or use a similar model as ours. For the protocols using the idea of sub-sampling clients, we denote the number of sampled clients as C which is sublinear in n. In Flamingo, the decryptor has an asymptotic cost slightly larger than n when dropouts happen.

Limitations. Flamingo assumes that the set of all N clients involved in a training session is fixed before the training starts and that in each iteration t some subset S_t from the N clients is chosen. This system does not naturally handle clients who dynamically join the training session. Armadillo allows clients to join dynamically, provided the clients register their public keys in the PKI beforehand. This is made possible because decryptor clients in Armadillo are stateless, unlike in Flamingo, where the decryptor clients hold shares of a secret key established at the beginning of the protocol.

Another unexplored aspect of this dissertation is handling an *adaptive* adversary that can dynamically change the set of parties that it compromises as the protocol executes. In BBGLR [BBG+20], the adversary can be adaptive across multiple iterations of aggregations but not within one aggregation; in our systems the adversary is static across all the iterations. To our knowledge, an adversary that can be adaptive within a single execution of aggregation has not been considered before in the federated learning setting. It is not clear that existing approaches from other fields [GHK+21] can be used due to different communication models.

Finally, secure aggregation reduces the leakage of individuals' inputs in federated learning but does not fully eliminate it. For example, the intermediate training results are leaked to the clients. It is important to understand what information continues to leak. A few recent works in this direction

are as follows. Elkordy et al. [EZE+23] utilize tools from information theory to bound the leakage with secure aggregation: they found that the amount of leakage reduces linearly with the number of clients. Wang et al. [WXWZ23] then show a new inference attack against federated learning systems that use secure aggregation in which they are able to obtain the proportion of different labels in the overall training data. While the scope of this attack is very limited, it may inspire more advanced attacks. So et al. [SAG+23] show that if the clients' inputs remain relatively stable and there is a significant overlap in the selected clients across iterations, then the server can infer an individual input by observing the sums over many training iterations. To mitigate this risk, they propose a batch partitioning strategy that groups clients into fixed batches, ensuring that a group of clients either participate together or not at all.

Pasquini et al. [PFA22] show an attack where a malicious server can circumvent secure aggregation by sending inconsistent models to clients. They suggest a mitigation for pairwise-masking based protocols (like Flamingo): binding the model hash to pairwise masks, which cancel out if all clients share the same hash. Armadillo does not rely on pairwise masking, but we can use a different approach: each client hashes the received model and sends the hash to the decryptors. The decryptors then perform a majority vote on the hashes and exclude shares from clients whose hashes do not match the majority.

CHAPTER 3

"PRIVATE PULL": PIR IN THE SHUFFLE MODEL

3.1 Introduction

A private information retrieval (PIR) protocol [CGKS95, KO97] allows a client to fetch an entry from a database server without revealing which entry was fetched. Specifically, the server holds a database $x = (x_1, \ldots, x_n)$ consisting of n bits (or generically, n symbols over an alphabet Σ) while the client holds an index $i \in \{1, \ldots, n\}$; the client wishes to obtain x_i while hiding i from the server.

PIR protocols have been broadly studied in two flavors—information-theoretic and computational—they each has their own merits. Information-theoretic protocols provide security against computationally unbounded adversaries and do not require "cryptographic" computation. Unfortunately, non-trivial information-theoretic PIR (i.e., less than n bits of communication) is impossible with only one server [CGKS95]. Consequently, PIR protocols in this setting need database replication across two or more non-colluding servers. This poses significant challenges for deployment: managing multiple databases is costly when they are large (e.g., data synchronization, replication cost), and enforcing non-collusion on the databases is hard in the real world—the database servers are often operated by a single company or organization in practice. On the other hand, computational PIR can work when only one server holds the database but only provides security against polynomial-time adversaries due to its reliance on cryptographic hardness assumptions. Yet, the cost of single-server computational PIR is usually high due to expensive cryptographic operations. Indeed, existing single-server protocols [AMBFK16, ACLS18, ALP$^+$21, MW22] are significantly slower than the multi-server information-theoretic ones [GCM$^+$16, GHPS22] (the only exceptions unfortunately require relatively large storage at clients [HHCG$^+$23, DPC23, ZSCM23]).

The shuffle model: PIR with many clients. How can we achieve the best of both worlds, under a possibly relaxed model? Recall that single-server information-theoretic solution is not possible in the standard model without using n bits of communication [CGKS95]. To circumvent this barrier in the standard model, Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS06] proposed a

relaxed model, where *many* clients (which can hold arbitrary query indices) simultaneously query a single server, but the clients are granted the ability to make anonymous queries to the server. Abstractly, we can think of the queries as being *shuffled* before reaching the server. It is important to note that the shuffling does not trivialize the PIR problem; it hides from the database who sends which message but not the message itself.

Consider a client using a *multi-server* PIR query algorithm to generate sub-queries for a query index. If these sub-queries were naively sent to a *single server*, the server would immediately learn the query index of this client. However, this work and [IKOS06] show the power of shuffling: if there are many clients and their sub-queries are randomly permuted by a shuffler before being sent to the server, then it is hard for the server—even one that is computationally unbounded as we show in this work—to figure out any of the client-query indices. Therefore, this single server in the shuffle model can simply perform "cheap" operations of the multi-server PIR scheme to answer sub-queries.

Understanding *the shuffle model* in the context of PIR is well-motivated by real-world applications: databases with high-volume queries, such as stock quotes and search engines, naturally enjoy the feature that thousands of users access the database at the same time, and therefore considering PIR with many simultaneously querying clients is sensible, particularly if it allows for lower server cost. Note that this is a substantially different goal from batch PIR [IKOS04, Hen16] which amortizes the cost of multiple queries from a single client (see Section 3.7). The shuffle model has been considered also in problems orthogonal to PIR, including secure aggregation [IKOS06, BBGN20, GMPV20] and differential privacy [BEM+17, CSU+19, EFM+20, CU21, AIVG22]. Analogously to these works, we view shuffling as an *atomic* operation; existing literatures on differential privacy [BBGN20] and anonymity [vdHLZZ15, LYK+19, APY20, DMS04, HSSN+22] discuss how to implement shuffling efficiently (see details in Section 3.1.2).

The shuffle PIR model opens a promising direction toward constructing efficient single-server PIR protocols. In this work, we establish the *theoretical feasibility of non-trivial single-server PIR with information-theoretic security in the shuffle model.*

3.1.1 Summary of contributions

Information-theoretic single-server PIR in the shuffle model. We present the first construction for single-server PIR in the shuffle model that has *sublinear communication* and *information-theoretic* security (with inverse-polynomial statistical error). Moreover, our construction is also *doubly efficient*: following one-time preprocessing on the server side, and without any state information on the client side, the server's per-query computation is sublinear in the database size.

Theorem 3.1.1 (Informal)**.** *For every constant $0 < \gamma < 1$, there exists a single-server PIR protocol in the shuffle model such that, on database of size n, and following one-time preprocessing on the server side, the protocol has $O(n^{\gamma})$ per-query computation and communication, and $O(n^{1+\gamma/2})$ server storage. This is achieved with the following information-theoretic security guarantee: for any inverse polynomial $\epsilon = 1/p_1(n)$, there exists a polynomial $p_2(n) = O(n^{1+4/\gamma} \cdot (p_1(n))^8)$ such that the protocol has ϵ-statistical security as long as the total number of queries made by (uncorrupted) clients is at least $p_2(n)$.*

As a key technique, we describe a generic *inner-outer* paradigm that composes together two standard (multi-server) PIR protocols: an outer and an inner layer, to build a PIR protocol in the shuffle model. Besides, our results are robust against imperfect shuffling/anonymity (Appendix C.2).

While the above protocol only achieves inverse-polynomial (rather than negligible) security error, this is in fact the standard notion of security in several important settings, including differential privacy [DN03, DMNS06], secure computation with partial fairness [Cle86, MNS09, GK10], and secure computation over one-way noisy communication [AIK$^+$21]. Our protocol demonstrates that *information-theoretic* security is indeed feasible *without database replication*. While concrete efficiency is not the focus of this work, our approach shows the unlocked efficiency for single-server PIR in the shuffle model, and a follow-up work has pushed this direction closer to practice [GIK$^+$24].

PIR with variable size database records. Real-world databases typically contain records in a variety of sizes, and revealing the record size typically reveals sensitive information about the record identity. Unfortunately, in the usual PIR setting, nothing better can be done apart from

padding all records to the same size and having clients retrieve the padded records; this poses an undue communication cost if the majority of clients only wish to retrieve small records.

By considering PIR in the shuffle model, we show how the size of any individual retrieved record can be hidden without any padding even where there are only a small number of querying clients. In particular, we show a novel approach for splitting a record of size ℓ to be retrieved such that the server only stores the records without padding as a database, and each client makes $\mathsf{polylog}(\ell)$ queries to this database; our construction only makes a *black-box* use of PIR protocols.

The problem of handling variable-sized records serves as an independent motivation for studying the shuffle model in the context of PIR.

3.1.2 Discussion on the shuffle model

Two-way shuffling. In this work, we assume a two-way shuffler, namely not only the clients can send messages anonymously to the server but also that the server can respond to clients without needing to know client's identities. This is a bit different from the differential privacy works in the shuffle model, where shuffled messages are delivered to a server for analytics and there is no communication back from the server to the clients. For this work, we view the two-way anonymous communication as an ideal primitive. Below we discuss how to realize this primitive in practice.

Comparing with the classical models. PIR in the shuffle model can also be viewed as a hybrid model between the standard single-server and multi-server PIR models: as an abstraction, the shuffler models a second "server" which is assumed to not collude with the main database server but does not hold a copy of the database and can only perform *database-irrelevant* computations. This alone makes the shuffle model interesting for practical deployments: non-collusion between two (or more) servers holding the same database can be difficult to enforce (since it is likely for them to be operated by the same company for data ownership reasons) making it a strong assumption in practice; in contrast, if only one server holds the database, then the "two" servers can be reasonably run by independent (and possibly geographically distributed) entities. We also note that it could be interesting to let this second database-irrelevant server perform more generic computations instead of just acting as a shuffler; and this is explored in our follow-up works [GIK+24].

In practice, this two-way shuffler can be realized in a distributed way [KEB98, Cha81, DMS04, BBGN20, BBG23]. If we wish to keep only a single entity as the shuffler, we can instantiate the two-way channel as follows. Let (pk_D, sk_D) be the public and secret key pair of the database server. For each message the client wants to send to the server, it encrypts the message using a fresh symmetric key and uses pk_D to encrypt the symmetric key. The client binds the two ciphertexts together and sends them to the shuffle server. The shuffle server learns nothing about the messages sent to the server, and the database server can decrypt each message—first using sk_D, then the symmetric key. The server then responds to each message with an answer encrypted under the symmetric key of that message. The shuffle server permutes all the encrypted answers and forwards them to the clients.

3.2 Preliminaries

Basic notation. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. $\mathbb{F}$ denotes a finite field. $\mathfrak{S}_c$ denotes the symmetric group containing all permutations of c elements. We use bold letters to denote vectors (e.g., $\mathbf{z}$). We use $\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2)$ to denote the statistical distance between the distributions $\mathcal{D}_1$ and $\mathcal{D}_2$.

Unless specified, logarithms are taken to the base 2. The notation $\mathsf{poly}(\cdot)$ refers to a fixed but unspecified polynomial in its parameter; we use $\mathsf{polylog}(\cdot)$ to mean $\mathsf{poly}(\log(\cdot))$. The notation $\widetilde{O}$ hides arbitrary polylogarithmic factors, i.e., $f(n) = \widetilde{O}(g(n))$ if $f(n) = O(g(n)) \cdot \mathsf{polylog} n$.

We use $\xrightarrow{\$}$ to denote uniformly random sampling, $\to$ for output by deterministic algorithms, and $\xrightarrow{\$}$ for output by randomized algorithms.

3.2.1 Multi-server PIR schemes

We now introduce the standard notion of multi-server information-theoretic PIR. We start with the basic definition below.

Definition 3.2.1 (PIR)**.** Let Σ be a finite alphabet. A k-server PIR protocol over Σ is a tuple $\Phi = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$ with the following syntax:

- $\mathsf{Setup}(x) \to P_x$: a deterministic algorithm executed by all servers that takes in an n-entry database $x \in \Sigma^n$ and outputs its encoding P_x.

- $\mathsf{Query}(i; n) \xrightarrow{\$} ((q_1, \ldots, q_k), \mathsf{st})$: a randomized algorithm (parameterized by n) executed by the client that takes in an index $i \in [n]$, and outputs sub-queries $q_1, \ldots, q_k$ and a state st. The sub-query q_ℓ is sent to the ℓ-th server.

- $\mathsf{Answer}^\ell(P_x, q_\ell) \to a_\ell$: a deterministic algorithm executed by the ℓ-th server that takes in the encoding P_x and a sub-query q_ℓ, and outputs an answer a_ℓ. Since the Answer algorithm may be different for different servers, we use ℓ to denote the algorithm used by server ℓ.

- $\mathsf{Recon}((a_1, \ldots, a_k), \mathsf{st}) \to x_i$: a deterministic algorithm executed by the client that takes in answers $a_1, \ldots, a_k$ (where a_ℓ is from the ℓ-th server) and the state st, and outputs $x_i \in \Sigma$.

Φ needs to satisfy the following correctness and security properties:

Correctness. For all $n \in \mathbb{N}$, any database $x = (x_1, \ldots, x_n) \in \Sigma^n$, and all $i \in [n]$,

$$\Pr\left[\begin{array}{rcl} & P_x & \leftarrow \quad \mathsf{Setup}(x) \\ \mathsf{Recon}((a_1, \ldots, a_k), \mathsf{st}) = x_i : \quad ((q_1, \ldots, q_k), \mathsf{st}) & \leftarrow^{\$} & \mathsf{Query}(i; n) \\ & (a_1, \ldots, a_k) & \leftarrow \quad (\mathsf{Answer}^\ell(P_x, q_\ell))_{\ell=1}^k \end{array}\right] = 1.$$

Intuitively, correctness says that the client always gets the correct value of x_i.

Security. For all $n \in \mathbb{N}$, $i \in [n]$, and $T \subset [k]$, define the distribution

$$\mathcal{D}_n(i, T) := \{\{q_\ell\}_{\ell \in T} : ((q_1, \ldots, q_k), \mathsf{st}) \leftarrow^{\$} \mathsf{Query}(i; n)\}.$$

We say that Φ has (t, ϵ)-privacy (where $t < k$, and $\epsilon = \epsilon(n)$), if for all $n \in \mathbb{N}$, any two indices

$i, i' \in [n]$, and any set $T \subset [k]$ such that $|T| < t$, we have

$$\mathsf{SD}(\mathcal{D}_n(i, T), \mathcal{D}_n(i', T)) \leq \epsilon(n).$$

Intuitively, (t, ϵ)-privacy says that any set of less than t colluding servers has a distinguishing advantage at most ϵ.

We now provide as background common PIR schemes that will be important later for our construction for PIR in the shuffle model. The constructions employ the following general outline: The servers encode the database $x \in \Sigma^n$ as a polynomial P_x. To query the database at position i, the client first encodes i into a vector $\mathbf{z}^{(i)}$ where the encoding is defined in a way that results in $P_x(\mathbf{z}^{(i)}) = x_i$. The client now evaluates P_x at $\mathbf{z}^{(i)}$ while hiding $\mathbf{z}^{(i)}$ from the servers: it secret shares $\mathbf{z}^{(i)}$ into k shares, and each share is sent to one of the k servers (through e.g., additive or Shamir sharing). Each server can then evaluate P_x on one share and send the result to the client, who is able to reconstruct the entry x_i.

Two-server PIR with additive shares

The first construction we describe is a PIR scheme from Beimel et al. [BIK05] which uses two non-colluding servers. Figure 3.2.1 contains the full description.

Setup. Consider a field $\mathbb{F}$ within which Σ can be encoded. The Setup algorithm encodes a database $x \in \Sigma^n$ into an m-variate polynomial $P_x \in \mathbb{F}[Z_1, \ldots, Z_m]$ as follows. First, choose m and $d < m$ such that $\binom{m}{d} \geq n$, and let $M = (M_1, \ldots M_n)$ denote a list of n monomials in the variables $Z_1, \ldots, Z_m$ with total degree exactly d and the degree of each variable at most 1. For simplicity, we pick the first n such monomials in lexicographic order of the variable indices (e.g., $Z_1Z_2Z_3$ appears before $Z_1Z_2Z_4$). The encoding P_x is now simply the linear combination $P_x = \sum_{i=1}^{n} x_i M_i$.[10]

Query. The Query algorithm starts by encoding the query index $i \in [n]$ into a binary vector $\mathbf{z}^{(i)} = (z_1^{(i)}, \ldots, z_m^{(i)}) \in \{0,1\}^m$ defined such that each $z_j^{(i)} = 1$ if and only if the monomial M_i

[10] One can choose a more complicated encoding in [BIK05] (E1 encoding scheme) that allows better parameters, namely $\sum_{\ell=0}^{d} \binom{m}{\ell} \geq n$.

contains the variable Z_j. Observe here that the Hamming weight of $\mathbf{z}^{(i)}$ is d since the monomials are also of degree d. Such encoding ensures that $P_x(\mathbf{z}_i) = x_i$. Then the sub-queries are generated by splitting $\mathbf{z}^{(i)}$ into two additive shares $\mathbf{z}_1^{(i)} = (z_{1,1}^{(i)}, \ldots, z_{m,1}^{(i)})$ and $\mathbf{z}_2^{(i)} = (z_{1,2}^{(i)}, \ldots, z_{m,2}^{(i)})$, i.e., $\mathbf{z}^{(i)} = \mathbf{z}_1^{(i)} + \mathbf{z}_2^{(i)}$. Here, $\mathbf{z}_\ell^{(i)}$ is sent to the ℓ-th server for $\ell = 1, 2$.

Answer. The Answer^ℓ algorithm run by the servers first views the database encoding P_x as a $2m$-variate polynomial P'_x defined as:

$$P'_x(Z_{1,1}, Z_{1,2}, \ldots, Z_{m,1}, Z_{m,2}) = P_x(Z_{1,1} + Z_{1,2}, \ldots, Z_{m,1} + Z_{m,2}).$$

Now, the ℓ^{th} server selects all the monomial terms in P'_x such that the number of $Z_{\_,\ell}$ (i.e., the variables where the second subscript is ℓ) is at least half of the variables in that term (in the exactly half case, the monomials are split between the two servers in a pre-determined way). Note that the total number of monomials in P'_x is $2^d \cdot n$, so there should be $2^{d-1} \cdot n$ monomials for each server. The ℓ-th server then evaluates its selected monomials at the point $\mathbf{z}_\ell^{(i)}$ and responds with the sum as the answer a_ℓ (which is now a polynomial in the remaining m variables). Further, observe that each monomial in P'_x is of degree d, and so after the server evaluation, the answer polynomial a_ℓ will be of degree at most $d/2$.

Reconstruction. Finally, given answer polynomials a_1, a_2, the client evaluates a_1 at $\mathbf{z}_2^{(i)}$ and a_2 at $\mathbf{z}_1^{(i)}$, and sums up the evaluation results in $\mathbb{F}$ to get $P_x(\mathbf{z}^{(i)}) = x_i$.

Cost. The parameters m and d can be chosen to be both $\Theta(\log n)$ such that $\binom{m}{d} \geq n$. In this case, the query size is $O(\log n)$ (since m elements in $\mathbb{F}_2$ are sent to each server) and the answer size is $O(\sqrt{n})$ (since specifying an m-variate polynomial of degree $d/2$ requires $\binom{m}{d/2} = O(\sqrt{n})$ terms).

k-server PIR with additive shares

The above protocol can also be generalized to k servers where the encoding $\mathbf{z}$ is now split into k additive shares. In this case, the servers express the m-variate degree-d polynomial P_x as km-variate

Let x be a database with size n and $\mathbb{F}$ be a field, where each entry x_i is in $\Sigma = \mathbb{F}$.

- $\mathsf{PIR.Setup}(x) \to P$:
  1. Choose m, d such that $\binom{m}{d} \geq n$.
  2. Let $M = (M_1, \ldots, M_n)$ be a list of n monomials in $\mathbb{F}[Z_1, \ldots, Z_m]$ with total degree d and intermediate degree at most 1. Sort all monomials that have m variables with degree d by a lexicographic order of the variables indices.
  3. Compute $P_x = \sum_{i=1}^{n} x_i M_i \in \mathbb{F}[Z_1, \ldots, Z_m]$.
  4. Compute a $2m$-variate degree-d polynomial P from P_x such that $P(Z_{1,1}, Z_{1,2}, \ldots, Z_{m,1}, Z_{m,2}) = P_x(Z_{1,1} + Z_{1,2}, \ldots, Z_{m,1} + Z_{m,2})$.
  5. Output P.
- $\mathsf{PIR.Query}(i; n) \to ((q_1, q_2), \mathsf{st})$, where $i \in [n]$:
  1. Let $\mathbf{z} = (z_1, \ldots, z_m)$ be the i-th binary vector such that $z_j = 1$ if and only if the monomial M_i contains the variable Z_j.
  2. Let $\mathbf{z}_1 \xleftarrow{\$} \mathbb{F}_2^m$, $\mathbf{z}_2 \leftarrow \mathbf{z} - \mathbf{z}_1$; and let $q_\ell \leftarrow \mathbf{z}_\ell$ for $\ell = 1, 2$. Set $\mathsf{st} = (\mathbf{z}_1, \mathbf{z}_2)$.
  3. Output $((q_1, q_2), \mathsf{st})$.
- $\mathsf{PIR.Answer}^\ell(P, q_\ell) \to a_\ell$ (for $\ell = 1, 2$):
  1. Let $\{M'_j\}_{j \in [2^m n]}$ be all monomials where the number $Z_{\_,\ell}$ is at least half of the variables.
  2. Output $a_\ell \leftarrow \sum_{j \in [2^m n]} M'_j(q_\ell)$.
- $\mathsf{PIR.Recon}((a_1, a_2), \mathsf{st}) \to x_i$:
  1. Parse st as $(\mathbf{z}_1, \mathbf{z}_2)$.
  2. Compute $x_i \leftarrow a_1(\mathbf{z}_2) + a_2(\mathbf{z}_1)$ (note that a_1 and a_2 are polynomials).
  3. Output x_i.

Construction 3.2.1: A two-server information-theoretic PIR [BIK05].

degree-d polynomial P'_x. That is,

$$P'_x(Z_{1,1}, \ldots, Z_{1,k}, \ldots, Z_{m,1}, \ldots, Z_{m,k}) = P_x(Z_{1,1} + \ldots + Z_{1,k}, \ldots, Z_{m,1} + Z_{m,k}).$$

Let $\mathcal{Z}_\ell$ be the set of monomials such that for each monomial, there are more $Z_{\_,\ell}$ than $Z_{\_,\ell'}$ for any $\ell' \neq \ell$. The set $\mathcal{Z}_\ell$ is assigned to the ℓ-th server. Moreover, the monomials in P'_x but not in any of $\mathcal{Z}_{\_,\ell}$'s will be divided to k servers in a pre-determined way. To issue a query for index i, the client encodes it as before to a binary string $\mathbf{z} \in \mathbb{F}_2^m$, and then splits it to k additive shares over $\mathbb{F}_2^m$, denoted as $\mathbf{z}_1, \ldots, \mathbf{z}_k$. The client sends to the ℓ-th server the share $\mathbf{z}_\ell$, and the server evaluates the assigned monomials using $\mathbf{z}_\ell$. The evaluation result is a polynomial of degree $(k-1)d/k$; this implies the answer size (which dominates the communication cost) is $O(n^{(k-1)/k})$.

Observe that using more additive shares gives worse efficiency but better privacy (since collusion

between any $k-1$ servers can be tolerated). Efficiency can be significantly improved to $O(n^{1/k})$ using CNF shares [ISN87] (instead of additive shares) where each server is now given a different $(k-1)$-sized subset of the additive shares. This is because the evaluation of P_x at $k-1$ shares results in an answer polynomial of degree at most $O(n^{1/k})$. The efficiency gain, however, comes at the cost of much stronger non-collusion assumption for PIR, namely that no two database servers can collude. Looking ahead, an interesting consequence of using the shuffle model is that our CNF-sharing based construction (Section 3.5.3) can significantly reduce communication *without* making any non-collusion assumptions on database servers (since there is only one database).

For simplicity, going forward, we will refer to the k-server PIR with additive shares as *k-additive* PIR and its CNF-variant as k-CNF PIR; we describe this in Figure 3.2.2.

k-server PIR with Shamir shares

In this section, we describe the k-server t-private PIR that uses Shamir secret sharing from [BIK05]. Full description is provided in Figure 3.2.3. We also call this the Reed-Muller PIR as it is closely related to Reed-Muller code.

Setup. Consider a field $\mathbb{F}$ within which Σ can be encoded. The Setup algorithm encodes a database $x \in \Sigma^n$ into a polynomial $P_x \in \mathbb{F}[Z_1, \ldots, Z_m]$ as follows: First, choose m and d such that $\binom{m+d}{d} \geq n$ and $|\mathbb{F}| > k > td$ (typically m, d, t are chosen first and then k and the field size $|\mathbb{F}|$ are deteremined accordingly). Let $\alpha_0, \ldots, \alpha_d$ be distinct elements in $\mathbb{F}$ (note that $d < |\mathbb{F}|$). The index i is encoded to the i-th vector $\mathbf{z}^{(i)}$ of the form $(\alpha_{\lambda_1}, \ldots, \alpha_{\lambda_m}) \in \mathbb{F}^m$ where $\sum_{j=1}^{m} \lambda_j \leq d$. There exists a set of polynomials $P^{(i)}(z_1, \ldots, z_m)$ of degree at most d such that $P^{(i)}(\mathbf{z}^{(i)}) = 1$ and $P^{(i)}(\mathbf{z}^{(j)}) = 0$ for all $i, j \in [n]$ and $i \neq j$. The full details of this encoding and the construction of $P^{(i)}$'s are provided in [BIK05, Appendix B].

Query. To generate the sub-queries, after encoding the index i to $\mathbf{z}^{(i)}$, the client first chooses m univariate polynomials $(R_1, \ldots, R_m) = R$ each of degree t such that $R(\mathbf{0}) = (R_1(0), \ldots, R_m(0)) = \mathbf{z}^{(i)}$. It then randomly picks $r_1, \ldots, r_k \in \mathbb{F}$ and computes the sub-query to be sent to the ℓ^{th} server as $q_\ell = R(r_\ell) \in \mathbb{F}^m$.

Let x be a database with size n, each entry x_i is in $\Sigma = \mathbb{F}$. There are s non-colluding servers.

- $\mathsf{PIR.Setup}(x) \to P$:
  1. Choose m, d such that $\binom{m}{d} \geq n$.
  2. Let $M = (M_1, \ldots, M_n)$ be a list of n monomials in $\mathbb{F}[Z_1, \ldots, Z_m]$ with total degree exactly d and intermediate degree at most 1. Sort all monomials that have m variables with degree d by a lexicographic order of the variables indices.
  3. Compute $P_x = \sum_{i=1}^{n} x_i M_i \in \mathbb{F}[Z_1, \ldots, Z_m]$.
  4. Compute a sm-variate degree-d polynomial P from P_x such that $P(Z_{1,1}, \ldots, Z_{1,s}, \ldots, Z_{m,1} \ldots Z_{m,s}) = P_x(Z_{1,1} + \ldots + Z_{1,s}, \ldots, Z_{m,1} + \ldots + Z_{m,s})$.
  5. Output P.
- $\mathsf{PIR.Query}(i; n) \to ((q_1, \ldots, q_s), \mathsf{st})$, where $i \in [n]$:
  1. Let $\mathbf{z} = (z_1, \ldots, z_m)$ be the i-th binary vector such that $z_j = 1$ if and only if the monomial M_i contains the variable Z_j.
  2. Let $\mathbf{z}_1, \ldots, \mathbf{z}_{s-1} \xleftarrow{\$} \mathbb{F}_2^m$ and $\mathbf{z}_s \leftarrow \mathbf{z} - \sum_{j=1}^{s-1} \mathbf{z}_j$.
  3. Let $q_\ell \leftarrow (\mathbf{z}_{\ell+1}, \ldots, \mathbf{z}_s, \mathbf{z}_1, \ldots, \mathbf{z}_{\ell-1})$ for $\ell \in [s]$. // cyclic shift
  4. Set $\mathsf{st} = (\mathbf{z}_1, \ldots, \mathbf{z}_s)$.
  5. Output $((q_1, \ldots, q_s), \mathsf{st})$.
- $\mathsf{PIR.Answer}^\ell(P, q_\ell) \to a$, for $\ell \in [s]$:
  1. Let $\{M'_j\}_{j \in [s^m n]}$ be all monomials pre-determined such that the number of $Z_{\_,\ell}$ is at most $1/s$ fraction.
  2. Output $a \leftarrow \sum_{j \in [s^d n]} M'_j(q_\ell)$.
- $\mathsf{PIR.Recon}((a_1, \ldots, a_\ell), \mathsf{st}) \to x_i$:
  1. Parse st as $(\mathbf{z}_1, \ldots, \mathbf{z}_s)$.
  2. Compute $x_i \leftarrow \sum_{\ell \in [s]} a_\ell(\mathbf{z}_1, \ldots, \mathbf{z}_\ell - 1, \mathbf{z}_{\ell+1}, \mathbf{z}_s)$.
  3. Output x_i.

Construction 3.2.2: An s-server PIR with CNF shares [BIK05]. Note that when $s = 2$, this is simply the 2-server additive PIR.

Answer. The Answer^ℓ algorithm evaluates P_x at q_ℓ and sends back $a_\ell = P_x(q_\ell)$. Note that the answer algorithm for this protocol is the same for all k servers.

Reconstruction. Finally, the Recon algorithm uses Lagrange interpolation on the points (r_1, a_1), …, (r_k, a_k) to compute a degree td polynomial $S = P_x \circ R$; the evaluation $S(0)$ will give the desired database entry x_i. This interpolation is possible when $k > td$ and $|\mathbb{F}| > k$.

Other notation. For a PIR protocol Φ, we use $\mathcal{E}_\Phi$ to denote the encoding space of all indices. We use $\mathcal{Q}_\Phi$ to denote the space of all possible sub-queries (note that $\mathcal{Q}_\Phi$ may not equal $\mathcal{E}_\Phi$). For example, in the two-server construction above, $\mathcal{E}_\Phi$ contains all binary strings with Hamming weight

Let $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ be a database.

$\mathsf{PIR.Setup}(x) \to P_x$:

1. Choose parameters m, d, k, t such that $\binom{m+d}{d} \geq n$ and $|\mathbb{F}| > k > td$.
2. Compute $P_x = \sum_{i=1}^{n} x_i P^{(i)}(z_1, \ldots, z_m)$, where $P^{(i)}(\mathsf{PIR.Enc}(i)) = 1$ and $P^{(i)}(\mathsf{PIR.Enc}(j)) = 0$ for all $i, j \in [n]$ and $i \neq j$.
3. Output P_x.

$\mathsf{PIR.Query}(i; n) \to ((q_1, \ldots, q_k), \mathsf{st})$, where $i \in [n]$:

1. Run $\mathsf{PIR.Enc}(i)$ and gets $\mathbf{z} \in \mathbb{F}^m$.
2. Choose a set of degree-t random polynomials $R = (R_1, \ldots, R_m)$ such that $R(\mathbf{0}) = \mathbf{z}$.
3. For $\ell \in [k]$:
   - Randomly choose r_ℓ from $\mathbb{F}$.
   - Set $q_\ell \leftarrow Q(r_\ell)$. Note that each $q_\ell \in \mathbb{F}^m$.
4. Set $\mathsf{st} = (r_1, \ldots, r_k)$.
5. Output $((q_1, \ldots, q_k), \mathsf{st})$.

$\mathsf{PIR.Answer}(P_x, q) \to a$:

1. Compute $a \leftarrow P_x(a)$.
2. Output a.

$\mathsf{PIR.Recon}((a_1, \ldots, a_k), \mathsf{st}) \to x_i$:

1. Parse $\mathsf{st} = (r_1, \ldots, r_k)$.
2. Interpolate a degree-td univariate polynomial $R \circ P_x$ from $\{(r_\ell, a_\ell)\}_{\ell=1}^{k}$.
3. Output $x_i \leftarrow (R \circ P_x)(0)$.

Construction 3.2.3: A k-server t-private PIR based on Reed-Muller code [BIK05].

d, and the space $\mathcal{Q}_\Phi$ is $\mathbb{F}_2^m$, i.e, in this case $\mathcal{E}_\Phi \subset \mathcal{Q}_\Phi$.

3.2.2 Balls and bins

We formulate the core analysis of our constructions using the widely-used balls-and-bins problem, which we provide background and notation for here. Abstractly, the balls-and-bins problem analyzes the distribution of B (identical) balls thrown into N bins according to some distribution D (often independent and uniformly at random). To denote a final configuration of balls, we use a N-length vector $\mathbf{u} = (u_0, \ldots, u_{N-1})$ where u_i denotes the number of balls in bin i. We say that $\mathbf{u} = (u_0, \ldots, u_{N-1})$ is (B, N)-valid if each $u_i \in \mathbb{Z}^{\geq 0}$ and $\sum_i u_i = B$. Since our analysis often deals with sharing over a group $\mathbb{G}$, we may also label the bins using elements from $\mathbb{G}$; when $\mathbb{G}$ is unspecified, it is taken to be $\mathbb{Z}_N$. In particular, we define the following:

Definition 3.2.2 (Valid configuration)**.** We say that a vector $\mathbf{u} = (u_0, \ldots, u_{N-1})$ is a valid (B, N) balls-and-bins configuration, or simply that $\mathbf{u}$ is (B, N)-valid if each $u_i \in \mathbb{Z}^{\geq 0}$ and $\sum_i u_i = B$.

Definition 3.2.3. Given (B, N)-valid configurations $\mathbf{u} = (u_0, \ldots, u_{N-1})$ and $\mathbf{v} = (v_0, \ldots, v_{N-1})$, we define the following useful terms:

- The *edit distance*, denoted by $\mathsf{ED}(\mathbf{u}, \mathbf{v})$ is defined as $\mathsf{ED}(\mathbf{u}, \mathbf{v}) = \frac{1}{2} \sum_{i=0}^{N-1} |u_i - v_i|$.

  Intuitively, this denotes the number of balls that need to be moved to convert $\mathbf{u}$ to $\mathbf{v}$. Note that the distance is symmetric since $\mathsf{ED}(\mathbf{u}, \mathbf{v}) = \mathsf{ED}(\mathbf{v}, \mathbf{u})$. The edit distance between two distributions $\mathcal{U}$ and $\mathcal{V}$, denoted by $\mathsf{ED}(\mathcal{U}, \mathcal{V})$; can now be defined as $\mathbb{E}_{\mathbf{u} \sim \mathcal{U}, \mathbf{v} \sim \mathcal{V}} [\mathsf{ED}(\mathbf{u}, \mathbf{v})]$.

- The *ball-intersection* $\mathbf{u} \sqcap \mathbf{v}$ is defined as $(c_0, \ldots, c_{N-1})$ where each $c_i = \min(u_i, v_i)$.

- The *ball-difference* $\mathbf{u} \ominus \mathbf{v}$ is defined as $(u'_0, \ldots, u'_{N-1})$ where each $u'_i = \max(0, u_i - v_i)$.

3.2.3 The "split and mix" technique

A core idea in our construction follows from an ingenious split-and-mix approach for secure summation by Ishai et al. [IKOS06]. Specifically, they give a one-round single-server secure aggregation protocol as follows: Each client *splits* its input into k additive shares; then, as part of the shuffle model, these shares from all the C clients are *mixed* together before being sent to the aggregation server who simply outputs the sum of all the shares. The security goal here is that server cannot infer anything about a particular client's input. More precisely, the shuffled shares of any two tuples of client inputs (with equal sum) should look indistinguishable. Ishai et al. [IKOS06] show that statistical security of $2^{-\sigma}$ can be achieved by using per-input $k = \Theta(\log C + \log p + \sigma)$ additive shares over a group of size p. Recent works [BBGN20, GMPV20] improve this bound to $k = \lceil 2 + \frac{2\sigma + \log_2(p)}{\log_2(C)} \rceil$ and show that at least 4 shares are necessary.

In our shuffle PIR context, we find that 2 additive shares are sufficient due to our query randomization technique and the usage of additional *noise* queries; this cannot be done in the summation setting as the final output could change. Towards reducing the communication of our PIR protocol, we also generalize the split-and-mix approach to CNF shares.

3.3 Definitions: PIR in the shuffle model

We now formally define single-server PIR in the shuffle model. The setting here is to consider a single server but many query-making clients while still retaining information-theoretic security. Importantly, we do not assume any coordination among clients.

Definition 3.3.1 (PIR in the shuffle model)**.** Let Σ be a finite alphabet. A (single-server) PIR protocol (over Σ) in the shuffle model is a tuple $\mathsf{ShPIR} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$ with a syntax similar to that of a k-server PIR (Definition 3.2.1) except for a few key changes. In particular:

- $\mathsf{Setup}(x) \to P_x$: a deterministic algorithm executed by the server that takes in an n-entry database $x \in \Sigma^n$ and outputs its encoding P_x.
- $\mathsf{Answer}(P_x, q_\ell) \to a_\ell$: a deterministic algorithm executed by the server that takes in the encoding P_x and a sub-query q_ℓ, and outputs an answer a_ℓ. Unlike in Definition 3.2.1, there is a single Answer algorithm.
- $\mathsf{Recon}(a_1, \ldots, a_k) \to x_i$: a deterministic algorithm executed by the client that takes in answers $a_1, \ldots, a_k$, where for all $\ell \in [k]$, a_ℓ is the answer to the client's sub-query q_ℓ; and outputs $x_i \in \Sigma$.

ShPIR needs to satisfy the following correctness property:

Correctness. For all $n \in \mathbb{N}$, database $x = (x_1, \ldots, x_n) \in \Sigma^n$, and $i \in [n]$,

$$\Pr\left[\mathsf{Recon}(a_1, \ldots, a_k) = x_i : \begin{array}{rcl} P_x & \leftarrow & \mathsf{Setup}(x) \\ (q_1, \ldots, q_k) & \leftarrow\!\!\$ & \mathsf{Query}(i; n) \\ (a_1, \ldots, a_k) & \leftarrow & (\mathsf{Answer}(P_x, q_\ell))_{\ell=1}^k \end{array}\right] = 1.$$

ShPIR also needs to satisfy the following security property in the model where client queries are shuffled before being sent to the server.

Security. We will parameterize security by a shuffler Π and a minimum number of honest client queries C. Formally, let $\Pi = \{\Pi_c\}_{c\in\mathbb{N}}$ be an ensemble such that Π_c is a distribution over the symmetric group $\mathfrak{S}_c$. When Π is unspecified, we assume that each Π_c is a uniform distribution over $\mathfrak{S}_c$; we refer to this as the uniform or perfect shuffler. We discuss imperfect shufflers in Appendix C.2.

For a given n, Π, and C, and given a tuple $I = (i_1, \ldots, i_C) \in [n]^C$ of client query indices, define the distribution

$$\widetilde{\mathcal{D}}_{n,\Pi,C}(I) = \left\{ \begin{array}{ll} & (q_1^{(1)}, \ldots, q_k^{(1)}) \leftarrow\!\!\$\ \mathsf{Query}(i_1; n) \\ & \ldots \\ \pi(\mathbf{q}): & (q_1^{(C)}, \ldots, q_k^{(C)}) \leftarrow\!\!\$\ \mathsf{Query}(i_C; n) \\ & \mathbf{q} \leftarrow (q_1^{(1)}, \ldots, q_k^{(1)}, \ldots, q_1^{(C)}, \ldots, q_k^{(C)}) \\ & \pi \xleftarrow{\$} \Pi_{kC} \end{array} \right\}.$$

Then, we say that ShPIR is (Π, C, ϵ)-secure if for every $n \in \mathbb{N}$ and all $C^* \geq C(n)$, and $I, I' \in [n]^{C^*}$, it holds that:

$$\mathsf{SD}(\widetilde{\mathcal{D}}_{n,\Pi,C^*}(I), \widetilde{\mathcal{D}}_{n,\Pi,C^*}(I')) \leq \epsilon(n).$$

Remark 5 (Randomized number of sub-queries). While Definition 3.3.1 considers a fixed number of sub-queries k, an interesting consequence of using the shuffle model is that it can support a variable k that is a randomized function of n. In the PIR construction, we will only use a fixed k in our main constructions. A variable k turns out to be useful for our analysis into PIR protocols that support variable sized records (see Section 3.6 for details).

Remark 6 (Number of queries v.s. number of clients). We allow clients to make multiple queries; since the queries are anonymous, the server cannot tell whether they are from the same client, and hence the security of PIR in the shuffle model actually relies only on the *total number of queries*, rather than the number of clients. Also, if we require some lower bound on the number of queries, we can let the clients (a given number of them) simply add more dummy queries for arbitrary indices to reach the bound. In formal statements we will always refer to the total number of queries, but

for ease of presentation, we may often implicitly assume that there are C clients that each make a single query.

Remark 7 (Adversarial clients). Our model also tolerates adversarial clients who collude with the server. As an extreme example, it is easy to see that if $C - 1$ clients collude with the server, then we are essentially back in the standard single-client setting.
Looking ahead though, our constructions will require a minimum number of *honest* client queries for security. If this is met, security is not reduced by any additional adversarial clients—even an unbounded number of them. Therefore, for simplicity, we can ignore these extra adversarial clients within our analysis.

Efficiency metrics. We measure the efficiency of PIR constructions in the shuffle model using a few metrics below. Since we consider many clients querying the server, we will characterize the cost per query.

- *Per-query (server) computation*: for answering each query, the number of bits that the server reads from the database and the preprocessing bits.
- *Per-query communication*: the sizes of the client query and the server response.
- *Server storage*: the total number of bits, including the preprocessing bits, that are stored by the server.
- *Message complexity*: for each query, the number of anonymous messages required to send. This is separately considered from the communication cost, since we need to take into account the anonymity cost. In particular, this will help us delineate between, e.g., sending one anonymous message of size s and sending s anonymous messages each of size 1 (since the latter may have more network overhead).

While our main focus is the server and the anonymity cost, we may also consider *per-query client computation*, which is the computational complexity for issuing each query and reconstructing the answer. One may also consider *client storage* which is omitted in this work as the clients in our

constructions are stateless.

3.4 Technical overview

In this section, we present a toy protocol, which is insecure but conveys our core ideas; we then outline the techniques for building our eventual protocol from the toy protocol.

An insecure toy protocol. The starting point is the classic two-server information-theoretic PIR scheme by Beimel et al. [BIK05]. In this scheme, a client first deterministically encodes its queried index $i \in [n]$ to a bit string $\mathbf{z}$ of length $m = O(\log n)$ (we call $\mathbf{z}$ the encoding of queried index, or simply query), and splits $\mathbf{z}$ to two *additive* shares in $\mathbb{F}_2^m$, $\mathbf{z}_1$ and $\mathbf{z}_2$ (we call them sub-queries), and then sends them to the two servers respectively.

We construct PIR in the shuffle model based on this protocol. Abstractly, each client generates two sub-queries (or shares) $\mathbf{z}_1$ and $\mathbf{z}_2$ as if it was querying using the above two-server scheme but in fact sends both sub-queries to a single server through an anonymous channel (which shuffles the sub-queries together with that from many other clients). Observe that this is exactly an instance of secure aggregation in the split-and-mix approach [IKOS06, BBGN20, GMPV20], where each input is split into two shares; the hope is that the server would learn nothing given the shuffled encoding shares from many clients.

There are two issues with this toy protocol. The first issue is obvious—the server learns the sum of all the encoding strings, and therefore can easily distinguish two sets of query indices by comparing the sum of their shares and the sum of their encodings. Note that leaking the sum to the server is exactly the goal of secure aggregation, but the sum should not be leaked in the PIR context. This leakage can be easily eliminated by letting one of the clients add a dummy share (a random string) to hide the sum. The second issue is more involved. In fact, splitting each input into only two shares is not enough to guarantee security; this can be demonstrated through a simple counter-example: suppose that the server wishes to distinguish between the 2-additive shares of zeros and that of ones (sharing over $\mathbb{F}_2$) . In the latter case, there is always an equal number of ones and zeros in the shares, while this is not true for the former case. This approach can be generalized to a "counting" based strategy (for sharing over any Abelian groups) and allows for

generic efficient distinguishing attacks [IKLM24]. While splitting into more additive shares, e.g., 4, is sufficient [BBGN20], this means we need a 4-server PIR (that has additive sub-queries) and thus leads to worse communication—$O(n^{3/4})$ in the 4-server scheme compared to $O(n^{1/2})$ in the two-server scheme (Section 3.2). On the road map to our general protocol with $O(n^{\gamma})$ communication (for any $\gamma > 0$), the first checkpoint is to bypass the above attack and achieve a protocol with $O(n^{1/2})$ communication; it turns out that the key ideas used for this also play a pivotal role in our final protocol design.

Randomizing inputs via the inner-outer paradigm. The core reason why the simple split-and-mix approach does not work with two additive shares is the presence of arbitrary correlation among the queries; indeed, if all queries were independent and uniformly random, then using two shares works perfectly. Our key insight to navigate around this is to randomize the queries using another PIR, resulting in *uniform random but pairwise independent* queries which is later shown to be sufficient for security.

Our construction employs a novel approach—the *inner-outer* paradigm, which composes a k-server PIR protocol as an *outer* layer with the previous 2-server PIR protocol (with 2-additive shares) as the inner layer. At a high level, the outer layer PIR randomizes the client queries before they get processed through the inner layer PIR. Below we call the outer layer protocol as OPIR and the inner layer protocol as IPIR.

Formally, the composition works as follows: for any database $x \in \{0,1\}^n$, on input an arbitrary query index $i \in [n]$, the client first runs the OPIR query algorithm to generate k queries $q_1, \ldots, q_k$; note that they naturally satisfy pairwise independence and each is uniformly random in the OPIR query space $\mathcal{Q}$, simply because of the security property of any PIR. Instead of sending them directly to the server, these queries are *interpreted as indices* to a new database x' of size $|\mathcal{Q}|$, where x' consists of the answers to all the possible OPIR queries (i.e., elements in $\mathcal{Q}$). Now the client runs IPIR query algorithm on the each of the k "indices" in $\{1, 2, \ldots, |\mathcal{Q}|\}$, and sends the IPIR sub-queries to the server. Specifically, the client maps an index to its encoding in the two-server protocol, and splits the encoding into 2 additive shares (sub-queries) in $\mathbb{F}_2^m$ where $m = O(\log |\mathcal{Q}|)$. Finally, to

have the compilation work, the server needs to build the database x' for IPIR in advance, which is feasible as long as $|\mathcal{Q}|$ is polynomial in n.

The upshot of this compilation is that the server now sees a set of shuffled shares generated from uniformly random and pairwise independent query indices to the database x'. As we shall show next, this randomization achieves that, for any two multi-sets of queried indices I, I' with distance at most δ, the resulting multi-sets after processing through OPIR will be J, J' will have distance in expectation $\sqrt{\delta}$, even though J, J' are larger than I, I'. The distance further decreases to $\sqrt[4]{\delta}$ after processing through IPIR (additive sharing). We will show that having each client add only one random noise sub-query (on top of its real sub-queries) is sufficient to hide the $\sqrt[4]{\delta}$ distance from the server.

Analyzing split-and-mix with pairwise independence. We now prove that the split-and-mix approach is sufficient once we have pairwise independent queries from the OPIR; we will use a ball-and-bins formulation for this analysis. The full details are given in Section 3.5.1.

Concretely, consider two arbitrary sets of client queries I and I'. Recall that each client query is first split into k uniformly random and pairwise-independent OPIR sub-queries; this is followed by 2-additive IPIR sharing, where each OPIR sub-query is further additively split into 2 shares in the IPIR space. Observe now that the OPIR queries of all clients can be viewed as throwing kC balls into $|\mathcal{Q}|$ bins where C is the number of clients and $\mathcal{Q}$ is the OPIR query space. Let $\mathcal{Y}(I)$ denote the distribution of the *balls-and-bins configuration* (i.e., a vector of random variables denoting the number of balls in each bin) of the OPIR sub-queries resultant from I. Next, given some configuration $\mathsf{y} \sim \mathcal{Y}(I)$, observe further that the 2-additive IPIR sharing can be viewed as creating a new, balls-and-bins configuration $\widetilde{\mathsf{y}}$ (now with $2kC$ balls), where each previous ball is now *split* into two balls; in particular, a ball in bin b within y results in two balls in random bins u and $b-u$ (where the bin labels are viewed as the additive group given by the IPIR query space). Denote the distribution of this new configuration by $\widetilde{\mathcal{Y}}(I)$. Roughly, the goal now is to prove that for arbitrary I and I', we can bound $\mathsf{SD}(\widetilde{\mathcal{Y}}(I), \widetilde{\mathcal{Y}}(I'))$ with some inverse polynomial in the number of clients C. This would imply that for an appropriately large $C = \mathsf{poly}(n)$, a server even with unbounded computation cannot

distinguish between two arbitrary sets of client PIR queries, except with probability that is inverse polynomial in n.

Looking ahead however, we will require some extra "noise" balls to be added uniformly at random, essentially to "smooth out" the distribution of $\widetilde{\mathsf{y}}$; in the PIR context, this corresponds to client sending an additional random IPIR share. Denote the balls-and-bins distribution of the shares with noise added as $\widetilde{\mathcal{Y}}^*(I)$. Our proof proceeds in the following three major steps:

For our first proof step, we focus on the OPIR and bound the expected *distance* in the balls-and-bins configuration for any two $\mathcal{Y}(I)$ and $\mathcal{Y}(I')$—intuitively, the distance here captures how many balls must be moved in one configuration to make it identical to the other. We show (Lemma 3.5.2), that the expected distance between the two configurations is bounded by $\sqrt{kC \cdot |\mathcal{Q}|/2}$. In other words, although I and I' can differ in the queries of all C clients, the expected distance between their OPIR queries is proportional to $\sqrt{C}$.

Now, for our second proof step, we analyze the additive splitting which takes place through the IPIR. Consider, in particular, two configurations y_1 and y_2 for the OPIR sub-queries. First, we show that it is sufficient to look only at the places where y_1 and y_2 differ in the context of the final statistical distance (see Lemma C.3.1); this allows us to on expectation, focus only on roughly $\sqrt{C}$ balls from the OPIR configurations when we later look at the IPIR sharing. We now show that when y_1 and y_2 have distance δ, post additive-sharing in the IPIR , the distance between the corresponding $\widetilde{\mathsf{y}}_1$ and $\widetilde{\mathsf{y}}_2$ reduces to $\Theta(\sqrt{\delta})$ (see Lemma C.3.2). Consequently, combining this with the first proof step implies that any sets of original client indices, once put through both the OPIR and IPIR, will have distance on expectation proportional to $\sqrt[4]{C}$.

The third and final proof step now shows that adding just 1 noise query per client results in being able to "hide" this $\sqrt[4]{C}$ difference in order to get $1/\mathsf{poly}(n)$ security; adding more noise queries improves the asymptotic bound on the number of clients needed to achieve the same security level. The analysis in this step roughly models the "toy in sand" problem—intuitively, how much "sand" (i.e., noise balls or queries) are needed to hide which bin a "toy" ball was initially put in.

Combining all the steps, we can show $\epsilon(n)$ statistical security as the total number of clients C is at least $\Omega(n^5/\epsilon^8)$; consequently, we get can any inverse-polynomial security where C is also polynomial in n. The (per-client) communication complexity for this construction is $O(n^{1/2})$.

Improving communication using CNF-shares. Following this, in Section 3.5.3, we show how a CNF-sharing based construction can be used as the IPIR to reduce the communication complexity; in particular, using an s-CNF sharing allows us to reduce the communication cost to $O(n^{1/s})$ given $\Omega(n^{2s+1}/\epsilon^8)$ clients for statistical security ϵ. This cleanly generalizes our earlier construction.

The security proof follows a similar outline as before but is somewhat more involved. We find a nice group theoretic formulation of the problem of understanding the symmetries within the CNF-sharing, which allows us to greatly simplify the analysis by leveraging simple results from that domain.

Impossibility results When considering PIR with multiple clients, it is useful to study the minimum number of clients required for security. After all, if we have a single client, then under the statistical security notion, this effectively means the client has $\Theta(n)$ communication. In Appendix C.1, we show that for any *linear* PIR (i.e., its encoding function is linear), which includes the constructions mentioned in Section 3.2 and others [BIK05, CGKS95], the number of clients required is at least the database size. Theorem C.1.1 shows that no linear PIR protocol in the shuffle model has statistical security better than $\frac{n-C}{n-1}$ for $C < n$.

3.5 A generic construction paradigm

We now present generic ways to build asymptotically efficient PIR protocols in the shuffle model from the PIR constructions mentioned in Section 3.2. The high-level idea is to compose together a protocol OPIR at the outer layer with a protocol IPIR at the inner layer, for randomizing the query indices. We call this inner-outer paradigm for constructing ShPIR protocols.

3.5.1 The main theorem

We start with our generic composition which uses an IPIR with two additive shares. Our main security result for this composition is given as Theorem 3.5.1. We provide an overview of the core

proof techniques in this section; the full proof is given in Appendix C.3.

Theorem 3.5.1 (ShPIR Composition Theorem for additive IPIR)**.** *Let Φ be any k-server t-private information-theoretic PIR scheme where $k > t > 2$; denote its sub-query space size by Q and its answer size by A. Let Ψ be 2-additive PIR defined in Construction 3.2.1. Then, for any database size $n \in \mathbb{N}$, given any $\epsilon > 0$, there exists a constant c_0 such that for $C \geq (c_0 Q^5)/(k\epsilon^8)$, the construction $\mathsf{ShPIR}(\Phi, \Psi)$ is a (Π, C, ϵ)-secure PIR in the shuffle model where Π is uniform. Here, Q, k, ϵ, C may all be functions of n. Furthermore, when $Q = \widetilde{O}(n)$ and assuming one-time preprocessing, the construction has:*

- *per-query server computation $O(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{1}{2}})$,*
- *per-query client computation $O(A \cdot k \cdot Q^{\frac{1}{2}})$,*
- *per-query communication $O(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{1}{2}})$,*
- *server storage $\widetilde{O}(A \cdot k^{\frac{3}{2}} \cdot Q^{\frac{3}{2}})$.*

Remark 8. In Theorem 3.5.1, we use C to implicitly mean the total number of queries from all *uncorrupted* clients. Furthermore, for any n-bit database, when ϵ is $1/p_1(n)$ for some polynomial p_1, C can be chosen as a polynomial $p_2(n) = c_0(Q(n))^5(p_1(n))^8/k(n)$ for some constant c_0. If C is exponentially large, we can get exponential security.

Remark 9 (Answering IPIR sub-queries). Recall that in the 2-additive PIR protocol (Figure 3.2.1), the servers respond to a client's sub-query knowing which server it acts for: after encoding the database as a $2m$-variate polynomial P'_x containing a set $\mathcal{M}$ of monomials, the first server evaluates only those monomials from a fixed set $\mathcal{M}_1$ at the client sub-query, while the second server evaluates monomials from the set $\mathcal{M}_2 = \mathcal{M} \setminus \mathcal{M}_1$. This means that it must be known to the servers whether they are the "first" or "second" server in the protocol. Consequently, when compiling this as the inner layer of our ShPIR construction, since there is only one server, it needs to figure out which shares to evaluate using $\mathcal{M}_1$ and which using $\mathcal{M}_2$.

One idea is to have the client label the shares; this significantly complicates the analysis since there

ShPIR Composition. A shuffle model PIR protocol $\mathsf{ShPIR}(\mathsf{OPIR}, \mathsf{IPIR})$ built using the inner-outer paradigm from a k-server OPIR, and a s-server IPIR is defined as follows:

- $\mathsf{ShPIR.Setup}(x) \to P$:
  1. Let $P_x \leftarrow \mathsf{OPIR.Setup}(x)$.
  2. Define a database x' of size n' as follows:
     - Let $n^* = |\mathcal{Q}_{\mathsf{OPIR}}|$ and let $L = (L_1, \ldots, L_{n^*})$ denote the sorting of the sub-query space $\mathcal{Q}_{\mathsf{OPIR}}$.
     - If the Answer algorithm is the same for all OPIR servers:
       For all $i \in [n^*]$, let $x'_i \leftarrow \mathsf{OPIR.Answer}(P_x, L_i)$.
       As a result, x' is of size $n' = n^*$.
     - If the Answer algorithm is different for the k OPIR servers:
       For $i \in [n^*], \ell \in [k]$: let $x'_{i+n' \cdot (\ell-1)} \leftarrow \mathsf{OPIR.Answer}^{\ell}(P_x, L_i)$.
       As a result, x' is of size $n' = kn^*$.
  3. Run $\mathsf{IPIR.Setup}(x')$ and output its result as P.
- $\mathsf{ShPIR.Query}(i; n) \to (q_1, \ldots, q_h)$, where $i \in [n]$ and $h = k(s+1)$:
  1. Initialize $(u_{\ell,j})_{\ell \in [k], j \in [s]}$.
  2. Let $(q'_1, \ldots, q'_k) \leftarrow\!\!\$\ \mathsf{OPIR.Query}(i; n)$.
  3. For $\ell \in [k]$,
     - If the Answer algorithm is the same for all k OPIR servers:
       Map q'_ℓ to the corresponding index $i'_\ell \in [n']$,
       i.e., $x_{i'_\ell} = \mathsf{OPIR.Answer}(P_x, q'_\ell)$.
     - If the Answer algorithm is different for the k OPIR servers:
       Map q'_ℓ to the corresponding index $i'_\ell \in [kn']$,
       i.e., $x_{i'_\ell} = \mathsf{OPIR.Answer}^{\ell}(P_x, q'_\ell)$.
     - Let $(\widetilde{q}_1, \ldots, \widetilde{q}_s) \leftarrow\!\!\$\ \mathsf{IPIR.Query}(i'_\ell; n')$.
     - Set $(u_{\ell,1}, \ldots, u_{\ell,s}) \leftarrow (\widetilde{q}_1, \ldots, \widetilde{q}_s)$.
  4. Let $(r_1, \ldots, r_k) \xleftarrow{\$} \mathcal{Q}_{\mathsf{OPIR}}$. // dummies
  5. Output $(u_{1,1}, \ldots, u_{k,s}, r_1, \ldots, r_k)$.
- $\mathsf{ShPIR.Answer}(P, q) \to a$:
  1. If IPIR has the same Answer algorithms for server, return $a = \mathsf{IPIR.Answer}(P, q)$;
     otherwise return
     $$a = \left\{ (\mathsf{IPIR.Answer}^{\ell}(P, q), \mathsf{label}\ \ell) \right\}_{\ell \in [s]}.$$
- $\mathsf{ShPIR.Recon}(a_1, \ldots, a_h) \to x_i$:
  1. Initialize $(v_{\ell,j})_{\ell \in [k], j \in [s]}$ and $(a'_\ell)_{\ell \in [k]}$.
  2. For $\ell \in [k]$, $j \in [s]$:
     - Let $a_{(\ell-1)\cdot k+j}$ be the answer to sub-query $q_{(\ell-1)\cdot k+j}$, namely $u_{\ell,j}$.
     - If IPIR has different Answer algorithms for the servers, parse $a_{(\ell-1)\cdot k+j}$ as
       $$\{(\widetilde{a}_1, \mathsf{label}\ 1), \ldots, (\widetilde{a}_s, \mathsf{label}\ s)\},\ \text{let } v_{\ell,j} := \widetilde{a}_j\ (\text{whose associated label is } j).$$
     - If IPIR has the same Answer algorithms for the servers, let $v_{\ell,j} = a_{(\ell-1)\cdot k+j}$.
  3. For $\ell \in [k]$:
     - $a'_\ell \leftarrow \mathsf{IPIR.Recon}(v_{\ell,1}, \ldots, v_{\ell,s})$.
  4. Output $x_i \leftarrow \mathsf{OPIR.Recon}(a'_1, \ldots, a'_k)$.

Construction 3.5.1: Composed ShPIR built using the inner-outer paradigm.

is now additional structure. Instead, we have the server answer each share twice: once according to $\mathcal{M}_1$, and once according to $\mathcal{M}_2$ and send back the tuple as its response. Since the client knows which was the first share and which was the second, it can pick the correct responses to be used for reconstruction. This only results in a $2\times$ blowup in the server communication. This is formally showed in Figure 3.5.1.

Remark 10 (Reduced cost for homogeneous servers). For similar reason as above, if OPIR has different Answer algorithms for the servers, the ShPIR server needs to store k sub-databases, where for ℓ-th sub-database the server treats $q \in \mathcal{Q}_{\mathsf{OPIR}}$ as the ℓ-th share and stores the corresponding answers. If $\mathsf{OPIR.Answer}$ is the same for all k servers, then ShPIR server only needs to store one such sub-database; as a result, both the per-query server computation and communication will be $O(A \cdot k \cdot Q^{\frac{1}{2}})$, and the server storage will be $O(A \cdot Q^{\frac{3}{2}})$. The client computation will be $O(A \cdot k \cdot Q^{\frac{1}{2}})$. See details in Appendix C.3.4.

3.5.2 Proof overview

Consider a client query index $i \in [n]$. Recall that our k-server OPIR will first encode i into the space $\mathcal{E}_{\mathsf{OPIR}}$ and then split it into k sub-queries in the space $\mathcal{Q}_{\mathsf{OPIR}}$. When composing with the IPIR, these k sub-queries will now be interpreted as IPIR query indices within the IPIR database of size $|\mathcal{Q}_{\mathsf{OPIR}}|$. Each of the k indices will now be encoded within the IPIR encoding space $\mathcal{E}_{\mathsf{IPIR}}$, and then split into 2 shares in the space $\mathcal{Q}_{\mathsf{IPIR}}$. Note that the space $\mathcal{Q}_{\mathsf{OPIR}}$ and $\mathcal{E}_{\mathsf{IPIR}}$ have the same size, which is the size of the IPIR database, and that $\mathcal{E}_{\mathsf{IPIR}} \subset \mathcal{Q}_{\mathsf{IPIR}}$. Going forward, for clarity, we keep using "sub-queries" for OPIR but use "shares" to mean the sub-queries for IPIR.

Given C clients, we will have kC total IPIR query indices encoded into $\mathcal{E}_{\mathsf{IPIR}}$; denote this by $\mathsf{y} = (y_1, \ldots, y_{kC})$ and let $\widetilde{\mathsf{y}}$ (of length $2kC$) denote its shares in $\mathcal{Q}_{\mathsf{IPIR}}$. Our main goal is to analyze the properties of $\widetilde{\mathsf{y}}$ since this will be the view of the server. In particular, given two lists of original query indices $I = (i_1, \ldots, i_C)$ and $I' = (i'_1, \ldots, i'_C)$, and their resulting shares $\widetilde{\mathsf{y}}$ and $\widetilde{\mathsf{y}}'$, we want to understand whether an adversary can find e.g., which of I or I' corresponds to $\widetilde{\mathsf{y}}$.

We now describe how to formulate our core analysis as a balls-and-bins problem. A key starting

observation here is that a uniformly random shuffler Π will eliminate any ordering within $\widetilde{\mathsf{y}}$ (and similarly for y). In turn, this allows us to essentially do our analysis using a balls-and-bins formulation, where each share in $\widetilde{\mathsf{y}}$ corresponds to a ball in one of $|\mathcal{Q}_{\mathsf{IPIR}}|$ bins. More precisely, the distribution of the shuffled shares in $\widetilde{\mathsf{y}}$ is exactly a $|\mathcal{Q}_{\mathsf{IPIR}}|$-dimensional distribution where the each component represents the distribution of the number of balls in that bin. Towards this, we also find it helpful to analyze y using a similar balls-and-bins formulation.

The crux of our analysis now boils down to quantifying the statistical distance between the distribution of balls over bins resultant from any two sets of original query indices I and I'. Specifically, define $\mathcal{Y}(I)$ to be the distribution of the balls-and-bins configuration of IPIR query indices y resultant from the original query indices I; define $\widetilde{\mathcal{Y}}(I)$ to be the distribution of its shares (i.e., corresponding to $\widetilde{\mathsf{y}}$). Roughly, the goal now is to show that for any I and I', we can bound $\mathsf{SD}(\widetilde{\mathcal{Y}}(I), \widetilde{\mathcal{Y}}(I'))$ with some inverse polynomial in the number of clients.

Looking ahead however, for our proof to go through, we will require some extra balls to be added uniformly at random, essentially to "smooth out" the distribution of $\widetilde{\mathsf{y}}$; this can also be thought of as uniformly random *noise*. In the PIR context, this effectively corresponds to each client sending a random sub-query in $\mathcal{Q}_{\mathsf{IPIR}}$. We denote the balls-and-bins distribution of the shares with noise added as $\widetilde{\mathcal{Y}}^*(I)$.

Remark 11 (Noise and communication complexity). We note that adding noise for each IPIR query index does not increase the asymptotic communication complexity for IPIR, i.e., the communication for an n-sized database is still $O(\sqrt{n})$. This is because the server will still evaluate each noise share either as the first or second share without changing the database encoding polynomial making the communication still $O(\sqrt{n})$. Note that adding noise is substantially different from splitting to more shares, i.e., if each IPIR index was instead split into more additive shares (corresponding to using an IPIR with more servers), then the number of variables in the encoding polynomial itself will be larger, which would increase the asymptotic communication.

Below we describe proof of the main theorem step by step. At a high level, we leverage balls-and-

bins style analyses to bound the statistical distance between $\widetilde{\mathcal{Y}}^*(I)$ and $\widetilde{\mathcal{Y}}^*(I')$. The rough idea will be to first compute the *edit distance* between the balls-and-bins configurations corresponding to the IPIR shares and then use that to bound the statistical distance after adding the random noise. Our proof proceeds in three major steps which we outline below.

Proof Step 1: *(Analyzing the edit distance of* OPIR *sub-queries; Appendix C.3.1).* Consider two lists of client indices $I = (i_1, \ldots, i_C)$ and $I' = (i'_1, \ldots, i'_C)$. Abstractly, the first part of our proof shows that the edit distance between the OPIR sub-queries generated from I and I' is *not too large.*

Recall that the t-out-of-k OPIR sub-queries generated are individually uniformly random, and are $(t-1)$-wise independent (and therefore also pairwise independent). Therefore, we can formulate our objective as the following balls-and-bins problem given in Lemma 3.5.2.

Lemma 3.5.2. *Suppose that B balls are thrown into N bins. Let $\mathcal{B}$ and $\mathcal{B}'$ be any two distributions of the final balls-and-bins configuration where each ball is thrown uniformly at random, and any two balls are independently thrown. Then:*

$$\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}[\mathsf{ED}(\mathbf{u},\mathbf{v})] \leq \sqrt{\frac{BN}{2}}.$$

Casting this result to our construction, since each client index generates k OPIR sub-queries and there are C clients in total, the expectation of edit distance (or differences) between any two sets of OPIR sub-queries (and consequently, the IPIR indices) is at most $\sqrt{kC\left|\mathcal{Q}_{\mathsf{OPIR}}\right|/2}$.

Proof Step 2: *(Analyzing the edit distance of 2-additive sharing in the* IPIR*; Appendix C.3.2).* Now that we have a bound on the edit distance between OPIR sub-queries (and consequently IPIR indices), our next step is to analyze the edit distance for shares in $\mathcal{Q}_{\mathsf{IPIR}}$. Recall that each encoded index in $\mathcal{E}_{\mathsf{IPIR}}$ is split into two additive shares. We model this as another balls-and-bins problem below.

Consider a (B, N)-valid configuration $\mathbf{u}$ and let $\mathsf{Share}_{\mathbf{u}}$ denote the distribution of randomly splitting each ball in $\mathbf{u}$ (in a group $\mathbb{G}$), i.e., for each ball b, throw one ball into a random bin $u \leftarrow\!\!\$\ \mathbb{G}$, and

another into bin $b-u$. The goal now is to bound the edit distance between $\mathsf{Share}_{\mathbf{u}}$ and $\mathsf{Share}_{\mathbf{v}}$ given the edit distance between $\mathbf{u}$ and $\mathbf{v}$.

To begin, we first show that in the context of the final statistical distance, it is sufficient to only consider the parts of $\mathbf{u}$ and $\mathbf{v}$ that are different. Let $\mathsf{Share}^{\ell}_{\mathbf{u}}$ denote the distribution of the balls-and-bins configuration when further throwing ℓ balls independently and uniformly at random following the sharing $\mathsf{Share}_{\mathbf{u}}$. In particular, we show (in Lemma C.3.1; Appendix C.3.2) that,

$$\mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}}, \mathsf{Share}^{\ell}_{\mathbf{v}}) \leq \mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}})$$

where $\ominus$ denotes the ball-difference operation defined in Section 3.2.2. Essentially, this will allow us to look at the splitting of only those balls that differ between $\mathbf{u}$ and $\mathbf{v}$; in particular, given (B, N)-valid $\mathbf{u}$ and $\mathbf{v}$ with edit distance δ, we will only need to concern ourselves with the (δ, N)-valid $\mathbf{u}' = \mathbf{u} \ominus \mathbf{v}$ and $\mathbf{v}' = \mathbf{v} \ominus \mathbf{u}$. We show the following result (in Lemma C.3.2; Appendix C.3.2):

$$\mathbb{E}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}'}, \mathsf{Share}_{\mathbf{v}'})\right] \leq \sqrt{2\delta N}.$$

Combining this with the result from the first proof part, we get:

$$\begin{aligned}\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}_{\mathbf{v}\ominus\mathbf{u}})\right] &\leq \sqrt{2N} \cdot \mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\sqrt{\mathsf{ED}(\mathbf{u},\mathbf{v})}\right] \\ &\leq \sqrt{2N}\left(BN/2\right)^{1/4} = (2)^{1/4}B^{1/4}(N)^{3/4}\end{aligned}$$

where the second step is by the concave Jensen's inequality.

Proof Step 3: *(Bounding the final statistical distance).* We are now ready to bound the final statistical distance between the final views of the server: $\widetilde{\mathcal{Y}}^*(I)$ and $\widetilde{\mathcal{Y}}^*(I')$. For this, we leverage a recent analysis by Boyle et al [BGIK22]. A straightforward corollary of their result can be abstractly stated as follows: Consider ℓ balls thrown independently and uniformly at random into N bins and let $\mathcal{U}_j$ denote the final distribution after another ball is added into bin j. Then for all bins j and

j', we have $\mathsf{SD}(\mathcal{U}_j, \mathcal{U}_{j'}) \leq \sqrt{N/\ell}$. Informally, this can also be thought of as a "toy in sand" problem of being able to hide the location (bin j or bin j') of an initial ball (i.e., the toy) after throwing in N random balls as noise (i.e., the sand). The same analysis can be extended to show that if there are Δ initial balls, after which ℓ random balls are thrown, the statistical distance will be bounded by $\Delta \cdot \sqrt{N/\ell}$. In the context of our PIR analysis, intuitively, Δ will represent the edit distance between $\mathsf{Share}_{\mathbf{u} \ominus \mathbf{v}}$ and $\mathsf{Share}_{\mathbf{v} \ominus \mathbf{u}}$, while the ℓ extra balls will represent the additional "noise" IPIR queries made. Note that when using this balls-and-bins analysis, we need to account for the fact that the edit distance is a distribution in our case, rather than a fixed number; it is straightforward to do so by using standard first-moment techniques (since we have a bound on the expectation).

Casting these analyses back to our PIR context, first notice that $\widetilde{\mathcal{Y}}^*(I)$ is nothing but the distribution $\mathsf{Share}^{\ell}_{\mathbf{u} \sim \mathcal{B}(I)}$ where $\mathcal{B}(I)$ is the distribution of OPIR sub-queries resulting from the indices I.

Looking ahead, we will use $\ell = kC$ uniformly random IPIR queries (i.e., k per client) as noise. A crucial point here is that the number of extra balls per client needs to be constant in C so that the individual communication complexity of each client does not depend on the how many clients are making queries. In fact, this also required our bound on the ED of the 2-additive sharing to be $o(\delta)$.

Combining the results from the previous parts, we show our main result:

$$\mathsf{SD}(\widetilde{\mathcal{Y}}^*(I), \widetilde{\mathcal{Y}}^*(I')) < \frac{3 \cdot N^{5/8}}{B^{1/8}} = \frac{3 \left|\mathcal{Q}_{\mathsf{IPIR}}\right|^{5/8}}{(kC)^{1/8}}.$$

since $N = |\mathcal{Q}_{\mathsf{IPIR}}|$ bins (query-space) and $B = kC$ balls (total sub-queries).

A final complication is bounding $|\mathcal{Q}_{\mathsf{IPIR}}|$ by Q (i.e., the size of OPIR sub-query space). We defer the details to Appendix C.5 (Lemma C.5.1); the high-level idea is that we let each IPIR database entry be A bits and consequently $|\mathcal{Q}_{\mathsf{IPIR}}|$ can be made $\widetilde{O}(Q)$. Then, assuming that there are $C = \Omega(n^{5+\nu}/k)$ client queries for some constant $\nu > 0$, the statistical distance can be bounded by some inverse polynomial $1/\mathsf{poly}(n)$ in n. More specifically, suppose that we wanted to bound the statistical distance by some inverse polynomial $\epsilon(n)$. Then, assuming at least $C(n) = \Omega(n^5/(k \cdot \epsilon^8))$ client queries, the statistical distance is bounded by ϵ. Consequently, the construction satisfies

(Π, C, ϵ)-security in the shuffle model where Π is the uniform shuffler.

Remark 12 (Concrete trade-off between the number of clients and the amount of noise). Recall that in the final step for bounding the statistical distance, we added kC balls in total, i.e., k independent random noise sub-queries for each client. We can, in fact, add just one random noise for each client and achieve the same level of security but at the cost of increasing the concrete number of clients required by a factor of k^2.

The reverse can be done as well; by adding more noise per client, say γk, the concrete number of clients can be reduced by a factor of γ^2 at the cost of increasing the communication of each client by a factor of γ (which would be asymptotically identical when γ is a constant). This is expected since intuitively noise sub-queries provide more randomness than arbitrary client queries.

3.5.3 Optimization

In this section, we describe how to generalize the IPIR to use CNF shares instead of additive shares. The upshot is that it allows us to reduce the communication complexity of the resultant ShPIR protocol to $O(n^c)$ for any constant $c > 0$.

Construction outline. Previously in Section 3.2.1, when looking at a standard multi-server PIR, we mentioned how using s additive shares instead of 2 results in an increased communication cost of $O(n^{(s-1)/s})$ but this can be reduced to $O(n^{1/s})$ at the cost of a stronger non-collusion assumption using a CNF sharing where each server is given a different $s-1$ sized subset of the additive shares. We show that the same strategy in fact also works in our inner-outer paradigm by using an IPIR with CNF-shares (the composed protocol is given in Figure 3.2.2). This compilation is particularly interesting since it requires *no extra non-collusion assumptions* to get the gain in efficiency (since the shuffle model already consists only of a single server). Instead, the trade-off will arise in the minimum number of clients required for security.

An s-CNF IPIR can simply be used as a drop-in replacement into Construction 3.5.1 to obtain a composed shuffle model protocol s-CNF-ShPIR. Here, upon obtaining the OPIR sub-queries, the client splits the encoding $\mathbf{z}$ into s additive shares $\mathbf{z}_1, \ldots, \mathbf{z}_s$ (in a group $\mathbb{G}$), and then constructs s

CNF shares where the i-th share is $(\mathbf{z}_{i+1}, \ldots, \mathbf{z}_s, \mathbf{z}_1, \ldots, \mathbf{z}_{i-1}) \in \mathbb{G}^{s-1}$ (see details in Figure 3.2.2). The CNF shares, i.e., sub-queries for IPIR, are then sent to the single server in s-CNF-ShPIR.

Theorem 3.5.3 shows the security and efficiency of this composition. We provide an outline of the proof in this section and defer the full proof to Appendix C.4.

Theorem 3.5.3 (ShPIR Composition Theorem for CNF IPIR)**.** *Let Φ be any k-server t-private information-theoretic PIR scheme where $k > t > 2$; denote its sub-query space size by Q and its answer size by A. Let Ψ be the s-CNF PIR defined in Construction 3.2.2. Then, for any database size $n \in \mathbb{N}$, and given any $\epsilon > 0$, there exists a constant c_0 such that for $C \geq (c_0 Q^{2s+1})/(k\epsilon^8)$, the construction $\mathsf{ShPIR}(\Phi, \Psi)$ is a (Π, C, ϵ)-secure PIR in the shuffle model where Π is uniform. Here, Q, k, ϵ, C may all be functions of n. Furthermore, when $Q = \widetilde{O}(n)$ and assuming one-time preprocessing, the construction has:*

- *per-query server computation $O(A \cdot k^{1+1/s} \cdot Q^{1/s})$,*
- *per-query client computation $O(A \cdot k \cdot Q^{1/s})$,*
- *per-query communication $O(A \cdot k^{1+1/s} \cdot Q^{1/s})$,*
- *server storage $\widetilde{O}(A \cdot k^{1+1/s} \cdot Q^{1+1/s})$,*

Similar to Remark 10, if OPIR.Answer *is the same for all k servers, then both the per-query server computation and communication will be $O(A \cdot k \cdot Q^{1/s})$, and the server storage will be $O(A \cdot Q^{1+1/s})$. The client computation will be $O(A \cdot k \cdot Q^{1/s})$.*

Proof outline. The overall structure of the proof is very similar to that of the composition theorem for additive IPIR; the main difference being in the second proof part to analyze the balls-and-bins distribution after the IPIR sharing which now involves CNF shares instead of additive shares.

Let s-CNF-Share$_{\mathbf{u}}$ be the distribution of the balls-and-bins configuration upon sharing each ball in $\mathbf{u}$ into s CNF shares in $\mathbb{G}^{s-1}$. Now, given (δ, N)-valid configurations $\mathbf{u}$ and $\mathbf{v}$, we want to bound the edit distance between s-CNF-Share$_{\mathbf{u}}$ and s-CNF-Share$_{\mathbf{v}}$; Through a natural group theoretic

formulation, this turns out essentially reduce to understanding the (cyclic rotational) symmetries of the CNF-sharing. Concretely, this allows us to show the following result (Lemma C.4.4):

$$\mathsf{ED}(s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}}, s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{v}}) \leq sN^{(s-1)/2}\sqrt{\delta}.$$

Notice that this bound nicely captures the bound on the edit distance correpsonding to Share in the 2-additive IPIR construction. Once we have this bound, the rest of the security proof of proceeds in exactly the same way as the one for Add-ShPIR. The full proof is given in Appendix C.4.

3.5.4 Concrete instantiations

We use Theorem 3.5.3 and instantiate OPIR with concrete protocols to derive our main results. To minimize the answer size, we use the k-server protocol with Shamir shares described in Section 3.2 (we interchangeably call it Reed-Muller code) to instantiate OPIR. To reduce the communication, we instantiate IPIR with the s-server CNF PIR protocol in Figure 3.2.2.

Parameters. We now discuss how to pick parameters for the composed PIR scheme that results in our main theorem. Now let OPIR be the k-server Reed-Muller PIR described in Section 3.2.1 (details in Figure 3.2.3); and IPIR be the 2-additive PIR (Section 3.2.1, Figure 3.2.1) or s-CNF PIR (Figure 3.2.2). Note that the 2-additive PIR is a special case of s-CNF PIR. Below we give a two-step overview of choosing parameters; Appendix C.5 gives a more fine-grained choice.

Let m, d, k, t be parameters for OPIR; recall that m is the number of variables, d is the polynomial degree, k is the number of OPIR servers and t is the privacy threshold. Let s, m', d' be parameters for IPIR, where s is the number of IPIR servers, m' is the number of variables and d' is the polynomial degree.

Step 1: *The* IPIR *database size resulting from* OPIR. Recall that IPIR database size n' depends on the size of sub-query space of OPIR. Now we show how to pick m, d, k, t for OPIR in order to get an $\Theta(n)$-sized IPIR database. We first choose m, d, t; and depending on them, choose $|\mathbb{F}|$ and k. The primary requirement is $\binom{m+d}{d} \geq n$ (see details in Section 3.2). Let m and t both be constants larger than 2, then the degree $d = O(n^{1/m})$. Secondly, we require that $|\mathbb{F}| > k > td$, and there exists k and

$|\mathbb{F}|$ that are $O(n^{1/m})$ that makes this requirement holds. Suppose $|\mathbb{F}| = c \cdot n^{1/m}$ for some constant c. The space $\mathcal{Q}_{\mathsf{OPIR}}$ is of size $|\mathbb{F}|^m = c^m \cdot n$, and since c, m are both constant, we have $|\mathcal{Q}_{\mathsf{OPIR}}| = \Theta(n)$. Let the $\mathbb{F}$ also be the field of database elements in IPIR (the field of OPIR), then the IPIR database consists of $n' = \Theta(n)$ entries with each entry of size $|\mathbb{F}| = \Theta(n^{1/m})$.

Step 2: *Preprocessing the* IPIR *database.* As we mentioned in Section 3.7, the server can pre-compute all the answers and store them as a lookup table. We now want to make the preprocessing for IPIR possible—we need to ensure the size of sub-query space $\mathcal{Q}_{\mathsf{IPIR}}$ is polynomial in n. First, from above, there exists a constant c' such that the IPIR database x' has size $n' = c' \cdot n$; and according to Section 3.2, there exists constants c'_1, c'_2 such that choosing $m' = c'_1 \log n$ and $d' = c'_2 \log n$ can ensure $\sum_{\ell=0}^{d'} \binom{m'}{\ell} \geq n$. Since each sub-query (CNF share) in IPIR is a vector with size $s-1$, therefore the size of $\mathcal{Q}_{\mathsf{IPIR}}$ (i.e., the number of entries in the lookup table) is $2^{m'(s-1)}$, which is $n^{c'_1(s-1)}$. When s is a constant, the server can pre-compute all answers to the sub-queries in polynomial time.

Each answer polynomial in IPIR has number of monomials $O(n^{1/s})$ where the coefficients of the monomials are in $|\mathbb{F}|$. Therefore, the number of bits of each answer is $\widetilde{O}(n^{1/s})$; the total number of preprocessing bits is bounded by $n^{c'_1 s}$.

Remark 13. Typically for Reed-Muller PIR (Figure 3.2.3), we choose the parameters m, d, k, t such that we can achieve polylogarithmic communication complexity with the minimum number of servers. However, in the inner-outer PIR composition, we want to make inner PIR database size $\Theta(n)$ so that we get $O(n^{1/s})$ communication, therefore we choose the number of servers to be $\Theta(n^{1/m})$, which is not as good as the typical case where there is polylogarithmic servers.

Theorem 3.5.4. *For every constant $0 < \gamma < 1$, there exists a Reed-Muller PIR Φ and a $(\lceil 2/\gamma \rceil)$-CNF PIR Ψ, such that on any database size $n \in \mathbb{N}$, given any $\epsilon > 0$, for all $C \geq c_0 n^{1+4/\gamma}/\epsilon^8$ where c_0 is some constant, the construction $\mathsf{ShPIR}(\Phi, \Psi)$ is a (Π, C, ϵ)-secure PIR where Π is uniform. Furthermore, assuming one-time preprocessing, the construction has*

- *per-query server computation $O(n^\gamma)$,*

- *per-query client computation* $O(n^\gamma)$,
- *per-query communication* $O(n^\gamma)$,
- *per-query message complexity* $O(n^\gamma)$,
- *server storage is* $\widetilde{O}(n^{1+\gamma/2})$.

We defer the full proof of Theorem 3.5.4 to Appendix C.5. One thing to note here is that the reduced communication per client with CNF shares comes at a price—to achieve the same level of security, we need a larger number of clients.

Remark 14 (Sub-polynomial communication assuming super-polynomial number of clients). An interesting consequence of the CNF-based IPIR is that it also enables more efficient protocols in the shuffle model. Using a $(\log n)$-server CNF-based protocol as our IPIR, we can achieve communication of $O(\mathsf{polylog}(n))$ with the assumption that there are at least some super-polynomial $n^{O(\log n)}$ number of clients. This results in better asymptotic complexity than the best existing protocols [DG15] in the standard-model PIR which use a constant numbers of servers. Note that the shuffle model compilation means that still only one server is required for our protocol and therefore we do not require the non-collusion assumptions of the standard-model CNF-based PIR.

Remark 15 (Negligible security with slightly sublinear communication). Our main result only achieves inverse-polynomial rather than negligible security error. We note that if one settles for *slightly sublinear* communication, there is a simple solution that achieves negligible security error and proceeds as follows. The server writes the n-bit database as an $m \times m$ matrix over $\mathbb{Z}_2$ where $m = \sqrt{n}$. Each client writes the column it is interested in as a unit vector $q \in \mathbb{Z}_2^m$. Assuming C clients query at the same time, where C is super-linear in n, each client splits the vector q into $k = O((m+\sigma)/\log C)$ additive shares, for security parameter $\sigma = \log^2 n$. For each query $q' \in \mathbb{Z}_2^m$, the server responds with $X \cdot q' \in \mathbb{Z}_2^m$. By the tight security analysis of the *additive* split-and-mix protocol [IKOS06, BBGN20, GMPV20], the security error is negligible in n, i.e., $\Theta(1/n^{\log n})$, and both the query and the answer are of size $k \cdot m = O(n/\log n)$.

3.6 PIR with variable-sized records

So far, our motivation for studying the shuffle model in the context of PIR was its natural occurrence in setting with multiple clients. In this section, we discuss a completely orthogonal use case—variable-sized database records—that also motivates our study of the shuffle model.

3.6.1 Motivation

Real-world databases typically contain records in a wide variety of sizes—a platform like Youtube, for instance, houses both 30-second shorts along with 5 hour long documentaries. In many situations, knowing merely the size of the record can help to identify the record itself (or at least substantially reduce the possibility space). Consequently, for PIR protocols to be implemented for such databases in practice, it is necessary to also prevent leakage of the size of the record accessed.

Unfortunately, existing literature on PIR usually considers databases where each record is of the same size. In particular, in the standard PIR setting, the only way to hide the size of the records retrieved is to pad each record to the length of the largest record (or some upper bound), or pack smaller records to the size of the largest record [GCM+16, ASA+21]. This means that a client who only wishes to access small records still needs to pay the high communication cost of retrieving the largest record. This is highly sub-optimal especially because in practice, we expect most clients to query for average-sized records, while only a few clients query for very large ones.

Contrary to the standard model, we show that the shuffle model enables more efficient hiding for the length of retrieved records (e.g., compared to padding every record to the maximum length). We give a way of private retrieval for variable-sized records without the need of any padding, where the cost of retrieving a record is proportional to the length of this record rather than the length of the largest record.

A simple yet effective solution in the shuffle model. A simple solution to handle variable-sized records in the shuffle model is to split each record of ℓ bits down into 1-bit sub-records, assuming 1 bit is the smallest length of the original records in the database w.l.o.g. Then, a client wanting to query for a record of length ℓ simply makes ℓ PIR queries. Note that given multiple clients, the shuffle

model provides the property to hide length of any individual queried record; only the total lengths of all retrieved records is leaked. This yet simple technique has a nice property: the communication per client is proportional only to ℓ but not the maximum record size L (which can be much larger).

It is easy to think that client anonymity trivializes the problem but this is not the case. The above simple solution works because the *multi-set* of all sub-lengths being queried is identical given a particular sum of record lengths. The same is not true if the records are split in other ways, e.g., using the powers of 2 corresponding to the binary encoding; there are many cases where a certain multi-set directly or indirectly reveals the set of original lengths configurations. Finally, revealing the total length of all queried records may leak non-trivial information when the number of clients is small. As a toy example, consider a database x with only two entries: x_1 of size 1, and x_2 of size 1000, and two clients in the shuffle model. In this case, the sum of the record lengths can directly reveal the queried indices of the two clients.

Our goal, therefore, is to identify under what parameter regimes the privacy of variable-sized records can be achieved, and if this can be done with better efficiency than the simple solution of splitting down to 1-bit sub-records.

3.6.2 Problem statement

Definition 3.6.1 (Record-splitting in the shuffle model)**.** A splitting algorithm $\mathsf{Split}(\ell; L)$, parameterized by a maximum length L, takes in length $\ell \in [L]$, and outputs a tuple $(w_j)_{j\in[h]}$ of values where $h = h(\ell, L)$ is a (possibly randomized) function of ℓ, L. The algorithm needs to satisfy the following properties:

Correctness. For every $\ell \in [L]$, letting $(w_1, \ldots, w_h) \leftarrow\!\!\$\ \mathsf{Split}(\ell; L)$, it holds that $\sum_{j=1}^{h} w_j \geq \ell$.

Security. As in our definition for PIR in the shuffle model, we will also parameterize security here by a shuffler Π and the number of clients C. Let $\Pi_c = \{\Pi\}_{c\in\mathbb{N}}$ be an ensemble such that Π_c is a distribution over the symmetric group $\mathfrak{S}_c$. When Π is unspecified, we assume that each Π_c is a uniform distribution over $\mathfrak{S}_c$; we refer to this as the uniform or perfect shuffler.

For a given L, Π, C, given a tuple $(\ell_1, \ldots, \ell_C) \in [L]^C$ of queried lengths, define the distribution

$$\mathcal{W}_{L,\Pi,C}(\ell_1, \ldots, \ell_C) = \left\{ \begin{array}{ll} & (w_1^{(1)}, \ldots, w_{h_1}^{(1)}) \leftarrow\!\!\$\ \mathsf{Split}(\ell_1; L) \\ & \ldots \\ \pi(\mathbf{w}): & (w_1^{(C)}, \ldots, w_{h_C}^{(C)}) \leftarrow\!\!\$\ \mathsf{Split}(\ell_C; L) \\ & \mathbf{w} \leftarrow (w_1^{(1)}, \ldots, w_{h_1}^{(1)}, \ldots, w_1^{(C)}, \ldots, w_{h_C}^{(C)}) \\ & \pi \xleftarrow{\$} \Pi_{(\sum_{j\in[C]} h_j)} \end{array} \right\}.$$

We say that Split is (Π, C, ϵ)-secure if for every $L \in \mathbb{N}$, on any two

$$(\ell_1, \ldots, \ell_C), (\ell'_1, \ldots, \ell'_C) \in [L]^C \text{ such that } \sum_{j\in[C]} \ell_j = \sum_{j\in[C]} \ell'_j,$$

it holds that $\mathsf{SD}(\mathcal{W}_{L,\Pi,C}(\ell_1, \ldots, \ell_C), \mathcal{W}_{L,\Pi,C}(\ell'_1, \ldots, \ell'_C)) \le \epsilon$.

Efficiency. We want to capture the efficiency of algorithm Split by measuring the portion of bits the client *wants* to retrieve in all the bits the client *actually* retrieves. We say that Split has *bit efficiency* μ, if for every $L \in \mathbb{N}$, and all $\ell \in [L]$, let $(w_1, \ldots, w_h) \leftarrow\!\!\$\ \mathsf{Split}(\ell; L)$, it holds that $(\sum_{j=1}^{h} w_j)/\ell \le 1/\mu$.

Note that $1/L \le \mu \le 1$. When $\mu = 1$, there is no waste on bits retrieved by the client; when $\mu = 1/L$, it is the same as padding.

Another efficiency metric we consider is *message complexity*, which we measure as $\mathbb{E}[h(\ell; L)]$, where the randomness comes from the Split algorithm.

Remark 16. In most situations, leaking merely the total lengths of all the queried records is quite benign. But if desired, the clients can mitigate this leakage by randomly increasing the queried length. Note that in the standard PIR, a single client query leaks an upper bound on the record size; while in the shuffle model, the size of any single queried item is not leaked.

PIR constructions with variable-sized records

The problem we consider has two parameters: the ratio of the maximum length to the minimum length, denoted as L; and the number of clients C. Assume w.l.o.g that the minimum length is 1, i.e., the maximum length is L. We first build some intuition on which parameter regimes support non-trivial solutions.

Parameter regime. Consider the following two cases, where 2^β clients query the database where the sum of lengths queried is $\beta \cdot 2^\beta$.

- Case 1: 2^j clients query the records with length $2^{\beta-j}$, for all $j = 1, \ldots, \beta$.
- Case 2: All the 2^β clients query the records with length β.

The above example conveys that, when the number of clients C is large compared to the record length, essentially there is no better solution than simply splitting each record down to one bit. Therefore, we will focus on solutions that work when C is much smaller than L, e.g., $C = \mathsf{polylog} L$.

Our idea to improve efficiency over the plain construction (i.e., split down to one bit) is to have the client retrieve more bits in a single PIR query. In particular, a client splits the length of its desired record according to some Split algorithm (specified later), and queries for the corresponding sub-records. We describe how the splitting compiles with PIR (in a black-box way) below.

Compiling record splitting with PIR. All the records are concatenated together to an n-bit string and the server stores this string as the database. Note that different from the standard model, the server does not pad the records to the same length. Then the server (virtually) prepares $\log n$ databases, each of n bits; the j-th database is partitioned into entries of size of 2^j for $j = 0, \ldots, \log n$. The server also sets up a helper database that stores, for each item, the range of indices in the original database (i.e., concatenation) that the record resides. For example, the 10-th entry in the helper database stores the information: the 10-th record consists of the 100 bits from index 200 to 300 in the n-bit database.

To retrieve an item, the client first makes a PIR query to the helper database to obtain the range of the indices to which the record corresponds. Suppose the record is of length ℓ. Now the client runs the algorithm Split that takes in ℓ and splits it to h sub-lengths. Then client retrieves h sub-records from the $\log n$ logical databases; these h sub-records should cover the entire range that the client wants to query.[11]

A concrete example is as follows. Suppose the client wants to retrieve the record that resides in range $[11, 16] \subset [n]$, and the Split algorithm outputs lengths 2, 4. The client will issue a query of the form $(\mathsf{PIR.Query}(6), \text{“size 2”})$ and $(\mathsf{PIR.Query}(4), \text{“size 4”})$. The former tuple queries the 6th size-2 sub-record, i.e., $[11, 12]$, and the latter tuple is for the 4th size-4 sub-record, i.e., $[13, 16]$. The server answers to the former tuple by viewing the n-bit database as 2 bits per entry, and answers to the latter tuple by viewing the database as 4 bits per entry.

3.6.3 Construction: Recursive splitting

Different from our earlier constructions in Section 3.5, Split relies on knowing the upper bound of the number of clients. We briefly describe the algorithm below and give the formal description in Figure 3.6.1.

Each client builds $\log L$ levels (or bins), where the j^{th} level represents length 2^j for $j = 0, \ldots, \log L$. The Split algorithm can be viewed as placing balls at the $\log L$ levels. For instance, two balls at level 0 and one ball at level 1 means the client should send two queries for length-1 sub-items and one query for a length-2 sub-item. The client starts with an initial configuration where ℓ is split into lengths of power of 2; this corresponds to placing at most one ball at the corresponding levels. For instance, if the record length is 40, the client splits it to 32 and 8, which corresponds to placing one ball at level 5 and one ball at level 3. Then starting from the highest level (the $(\log L)$-th level), for each ball at the current level j, with probability $1/2$ split it to two balls and place them at level $j - 1$; with probability $1/2$ the client leaves this ball at the current level. The eventual balls-to-bins configuration is the output of Split algorithm.

[11] The client can label the query with the level number so that the server knows how to partition the database when answering queries.

Parameters.
L: Maximum length of database records (w.l.o.g. assume L is power of 2)
ρ: Full-split depth $\rho \in [\log L]$.
Inputs.
ℓ: the length of query record.

1. Let $B_0, \ldots, B_{\log L - 1} \in \{0,1\}$ be binary decomposition of ℓ, i.e., $\ell = \sum_{j=0}^{\log L - 1} B_j \cdot 2^j$.
2. For j from $\log L - 1$ to $\log L - \rho$: // fully split
   (a) Set $B_{j-1} \leftarrow 2B_j$.
3. For j from $\log L - \rho$ to 1: // for each sub-record, split with probability 1/2
   (a) Set $S_j := 0$.
   (b) For $b \in [B_j]$: Sample $r \xleftarrow{\$} \{0,1\}$; if $r = 0$, then set $S_j \leftarrow S_j + 1$.
   (c) Set $B_j \leftarrow B_j - S_j$ and set $B_{j-1} \leftarrow B_{j-1} + 2S_j$.
4. Output $(\underbrace{2^0, \ldots, 2^0}_{B_0}, \underbrace{2^1, \ldots, 2^1}_{B_1}, \ldots, \underbrace{2^{\log L-1}, \ldots, 2^{\log L-1}}_{B_{\log L-1}})$.

Construction 3.6.1: Algorithm Split.

In our construction, there is a parameter ρ that specifies the depth of the levels until which all balls are split. ρ is a trade-off factor between efficiency and security: when $\rho = \log L$, we get perfect security with message complexity equal to the length of the record; when $\rho = 0$, no ball is split and the message complexity is simply 1 but without security. Theorem 3.6.2 gives the property of our construction; the full proof is deferred to Appendix C.6.

Theorem 3.6.2. *Given an upper bound C on the number of clients, a uniform Π, and any $\epsilon > 0$, there exists $\rho = \rho(C, L, \epsilon)$ such that the algorithm* Split *in Construction 3.6.1 is (Π, C, ϵ)-secure with bit efficiency $\mu = 1$ and message complexity $(2C \log \ell)/\epsilon^2$ for retrieving record of length ℓ. Asymptotically, when $C = \mathsf{polylog} L$ where L is the maximum length of records, the message complexity for a client retrieving a record of size ℓ is $\mathsf{polylog}\ell/\epsilon^2$.*

3.7 Related work and discussion

Below we survey existing work in the standard PIR model and contrast them with this work.

Kushilevitz and Ostrovsky [KO97] gave the first single-server PIR scheme based on quadratic residuosity assumption with linear server computation. Beimel, Ishai and Malkin [BIM00] later showed that linear server computation (no matter in multi-server or single-server schemes) is inherent if no

extra storage (at the server or the clients) is allowed. Subsequently, there are many works aiming to construct PIR with fast server computation under different models, and we categorize them as follows.

PIR with batched queries. "Batch PIR" processes a batch of queries to achieve sublinear cost per query. The batch PIR schemes [IKOS04, Hen16, CHLR18, ACLS18, MR23] work in the standard PIR model but amortize costs when a *single* client wants to simultaneously query multiple records, and they require the client to know the sequence of queries it makes in advance (i.e., the batch is non-adaptive). In contrast, we consider a fundamentally different model, where multiple clients simultaneously query a single server and their queries are shuffled. A qualitative advantage of the shuffle model is that it enables sublinear computation in the single-server setting without using cryptographic assumptions.

PIR with preprocessing. This class of schemes [BIM00, WY05, CHR17, BIPW17, Lip09, LMW23] utilize extra storage at the server(s), where the database is encoded into some (larger) forms that allow answering each query with sublinear computation. Our schemes fall in this category, where the server pre-computes the answers to all the sub-queries and stores them in a table, and the client performs a sublinear number of table lookups (in a private way). This is feasible (polynomial time in n) in terms of preprocessing cost and server storage when each sub-query has bit length $O(\log n)$.

PIR in the offline/online model. Recent work [CGHK22] construct PIR schemes where the extra storage is incurred at the clients: a client first runs an offline phase with the server, where the server does a (super)linear computation to generate hints and sends them to the client. In the subsequent online phase, the server answers each query with sublinear cost and the client obtains the queried item with the help of the hints. The hints can be reused for $n^{1/4}$ number of online queries, namely, the offline cost is amortized over an adaptive batch of queries.

There are also schemes [HHCG$^+$23, DPC23, ZPSZ23] that work in a slightly different offline/online model, where they allow $\mathsf{poly}(n)$ number of online queries (from the same client) but with linear (though concretely fast) server computation per query [HHCG$^+$23, DPC23]; or they have linear

client communication in the offline phase in exchange for sublinear online computation [ZPSZ23]. Our schemes do not require extra hints at clients, and the client can make an unlimited number of queries after a one-time server preprocessing.

Other works towards doubly efficient PIR. Ishai et al. [IKOS06] give a single-server PIR construction in the shuffle model as follows: each client runs a query algorithm of an information-theoretic multi-server PIR [BIK05] on its query index and generates sub-queries, following which the sub-queries from many clients are shuffled together and sent to the server. However, this construction is shown to be secure only under non-standard computational assumptions (specifically the hardness of reconstructing noisy low-degree curves in a low-dimensional space [IKOS06, CS03]). In contrast, our construction provides statistical security. The trade-off is that we require more clients accessing the database compared to theirs, and settle for inverse-polynomial security error. We leave open the possibility of eliminating these limitations. An alternative avenue for future work is obtaining more efficient *computational* PIR schemes in the shuffle model, improving the efficiency of the shuffle-based protocol from [IKOS06]. While there are single-server doubly efficient computational PIR protocols even in the plain model [BIPW17, CHR17, LMW23], they either rely on obfuscation or fully homomorphic encryption and therefore the potential of achieving good concrete efficiency along this direction may be limited. The shuffle model seems to have better potential for practical single-server solutions.

Differential privacy (DP) for PIR. A line of work [TDG16, AIVG22] considers the DP notion for PIR assuming client anonymity. Here, clients send their query indices via onion routing to the server, and privacy is guaranteed by the shuffling of client indices along with some noise queries. Here DP guarantees that the server cannot distinguish neighboring sets of queries (i.e., differing in exactly one client). Unfortunately, DP is substantially *weaker* than standard PIR security and therefore insufficient in any application where client queries can be *arbitrarily correlated*, as we illustrate below.

Consider an example where a disproportionately large number of clients access the same sensitive entry in the database; in standard security notion, this should be indistinguishable (statistically

or computationally) from the case where nobody accesses this entry, whereas the DP solutions fail here. Meanwhile, in many applications, the information of whether clients are correlated accessing the same or similar items can give the adversaries extra powers: consider a cloud service that stores encrypted medical records, and the adversary can use the access frequencies of the records together with public health statistics to recover the underlying plaintext of the encrypted records [ZKP16, GKL$^+$20]. In contrast, our construction achieves a stronger notion: for any set of query indices, the messages observed by the server look close to random.

Future research. While the per-query cost in this work is only $O(n^\gamma)$ for any small $\gamma > 0$, it requires a large number of clients—specifically, $n^{1+4/\gamma}$—to access the database simultaneously. An interesting direction for future research is obtaining concretely efficient PIR schemes in the shuffle model, possibly by settling for computational security. A first step of this direction was taken recently in our follow-up work [GIK$^+$24].

We show that shuffling sub-queries of a simple k-server PIR protocol suffices to achieve privacy. In this PIR protocol, the server represents the n-bit database as a $\sqrt{n} \times \sqrt{n}$ matrix, and a client queries the i-th column of the database that contains their desired bit. The query is represented as one-hot vector of length $\sqrt{n}$ where the i-th position is 1. To generate sub-queries, a client additively splits the one-hot vector into k shares, and each share is the sub-query to a server. Each server multiplies the database matrix with the received sub-query and gives back the answer to the client. The client can reconstruct a column of the database that contains their desired bit. The privacy from shuffling these sub-queries reduces to a hardness assumption called *multi-disjoint syndrome decoding*, a generalization of the classical syndrome decoding assumption [BFKL93, AIK07].

APPENDIX A

DEFERRED MATERIALS FOR FLAMINGO PROTOCOL

A.1 Failure and threat model details

In this section, we give the full details of the dropout rate and the corruption rate (§2.2.3).

Dropout rate. Recall that we upper bound the dropout rate of the sum contributors (S_t) in one round as δ. For decryptors, we consider the dropout rate in one summation round and assume it is at most δ_D. Note that δ and δ_D are individually determined by the server timeout at those steps (recall that in each round, clients in S_t only participate in the first step; the following two steps only involve decryptors).

Corruption rate. For corruption, we denote the corrupted rate in S_t as η_{S_t} and the corrupted rate in decryptors as η_D. In the Flamingo system, η is given; η_{S_t} and η_D depends on η. Note that the fraction of malicious clients in a chosen subset of $[N]$ (e.g., S_t, $\mathcal{D}$) may not be exactly η, but rather a random variable η^* from a distribution that is parameterized by η, N and the size of the chosen set. Since the expectation of η^* is equal to η, and when the size of the chosen set is large (e.g., S_t), the deviation of η^* from η is negligible (i.e., η^* is almost equal to η). Therefore, η_{S_t} can be considered as equal to η. On the other hand, since $\mathcal{D}$ is a small set, we cannot assume η_D is equal to η. Later in Appendix A.3 we show how to choose L to ensure η_D satisfies the inequality required in Theorem 2.2.4 with overwhelming probability.

Security parameters. In the following definitions and proofs, we use κ for the information-theoretic security parameter and λ for the computational security parameter.

A.2 Deferred details for full protocol

A.2.1 Definition of cryptographic primitives

In this section, we formally define the cryptographic primitives used in Flamingo protocol that are not given in Section 2.2.4.

Definition A.2.1 (DDH assumption)**.** Given a cyclic group $\mathbb{G}$ with order q, and let the generator

Protocol Π_{setup}.

Parties. Clients $1, \ldots, N$ and a server.

Parameters. Number of pre-selected decryptors L. Let $L = 3\ell + 1$.

Protocol outputs. A set of t clients ($2\ell + 1 \leq t \leq 3\ell + 1$) hold secret sharing of a secret key SK. All the clients in $[N]$ and the server hold the associated public key PK.

- The server and all the clients in $[N]$ invoke $\mathcal{F}_{\text{rand}}$ and receive a binary string $v \xleftarrow{\$} \{0,1\}^\lambda$.
- The server and all the clients in $[N]$ computes $\mathcal{D}_0 \leftarrow \textsc{ChooseSet}(v, 0, L, N)$.
- All the clients $u \in \mathcal{D}_0$ and the server run Π_{DKG} (Fig. A.2).
- The server broadcasts the signed PKs received from the clients in $\mathcal{D}_0$ to all the clients in $[N]$.
- A client in $[N]$ aborts if it received less than $2\ell + 1$ valid signatures on PKs signed by the parties defined by $\textsc{ChooseSet}(v, 0, L, N)$.

Figure A.1: Setup phase with total number of clients N. $\mathcal{F}_{\text{rand}}$ is modeled as a random beacon service.

of $\mathbb{G}$ be g. Let a, b, c be uniformly sampled elements from $\mathbb{Z}_q$. We say that DDH is hard if the two distributions (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) are computationally indistinguishable.

Definition A.2.2 (ElGamal encryption)**.** Let $\mathbb{G}$ be a group of order q in which DDH is hard. ElGamal encryption scheme consists of the following three algorithms.

- $\text{AsymGen}(1^\lambda) \to (SK, PK)$: sample a random element s from $\mathbb{Z}_q$, and output $SK = s$ and $PK = g^s$.
- $\text{AsymEnc}(PK, h) \to (c_0, c_1)$: sample a random element y from $\mathbb{Z}_q$ and compute $c_0 = g^y$ and $c_1 = h \cdot PK^y$.
- $\text{AsymDec}(SK, (c_0, c_1)) \to h$: compute $h = (c_0^{SK})^{-1} \cdot c_1$.

We say that ElGamal encryption is secure if it has CPA security. Note that if DDH assumption (Def.A.2.1) holds, then ElGamal encryption is secure.

Definition A.2.3 (Authenticated encryption)**.** An authenticated encryption scheme consists of the following algorithms:

- SymAuthGen$(1^\lambda) \to k$: sample a key k uniformly random from $\{0,1\}^\lambda$.
- SymAuthEnc$(k, m) \to c$: take in a key k and a message m, output a ciphertext c.
- SymAuthDec(k, c): take in a key k and a ciphertext c, output a plaintext m or $\perp$ (decryption fails).

We say that the scheme is secure if it has CPA security and ciphertext integrity.

For simplicity, we use AsymEnc and SymAuthEnc to refer to the encryption schemes.

Definition A.2.4 (Signature scheme)**.** A signature scheme consists of the following algorithms:

- $SGen(1^\lambda) \to (sk, pk)$: generate a pair of siging key sk and verfication key pk.
- $Sign(sk, m) \to \sigma$: take in a signing key sk and message m, outputs a signature σ.
- $VerSig(pk, m, \sigma) \to b$: take in a verification key pk, a messagee m and a signature σ, output valid or not as $b = 1, 0$.

We say that the signature scheme is secure if the probability that, given $m_1, \ldots, m_z$, an attacker who can query the signing challenger and finds a valid (m', σ') where $m' \notin \{m_1, \ldots, m_z\}$ is negligible.

A.2.2 Setup phase and distributed key generation

The setup protocol is conceptually simple, as shown in Figure A.1. A crucial part of the setup phase is the distributed key generation (DKG). We first describe the algorithms used in DKG.

Algorithms. Let $\mathbb{G}$ be a group with order q in which discrete log is hard. The discrete-log based DKG protocol builds on Feldman verifiable secret sharing [Fel87], which we provide below. The sharing algorithm takes in the threshold parameters L, ℓ, and a secret $s \in \mathbb{Z}_q$, chooses a polynomial with random coefficients except the constant term, i.e., $p(X) = a_0 + a_1 X + \ldots + a_\ell X^\ell (a_0 = s)$, and outputs the commitments $A_k = g^{a_k} \in \mathbb{G}$ for $k = 0, 1, \ldots, \ell$. The j-th share s_j is $p(j)$ for $j = 1, \ldots, L$.

To verify the j-th share against the commitments, the verification algorithm takes in s_j and a set

of commitments $\{A_k\}_{k=0}^{\ell}$, and checks if

$$g^{s_j} = \prod_{k=0}^{\ell} (A_k)^{j^k}.$$

We define the above algorithms as

- $FShare(s, \ell, L) \rightarrow \{s_j\}_{j=1}^{L}, \{A_k\}_{k=0}^{\ell}$,
- $FVerify(s_j, \{A_k\}_{k=0}^{\ell}) \rightarrow b$ where $b \in \{0, 1\}$.

The GJKR-DKG uses a variant of the above algorithm, $PShare$ and $PVerify$ based on Pedersen commitment, for security reason [GJKR06]. The $PShare$ algorithm chooses two random polynomials

$$p(X) = a_0 + a_1 X + \ldots + a_\ell X^\ell, \quad a_0 = s$$

$$p'(X) = b_0 + b_1 X + \ldots + b_\ell X^\ell$$

and outputs

$$\{p(j)\}_{j=1}^{L}, \{p'(j)\}_{j=1}^{L}, C_k := g^{a_k} h^{b_k} \text{ for } k = 0, \ldots, \ell,$$

where $g, h \in \mathbb{G}$.

To verify against the share $s_j = p(j)$, $PVerify$ takes in $s'_j = p'(j)$ and $\{C_k\}_{k=0}^{\ell}$, and checks if

$$g^{s_j} h^{s'_j} = \prod_{k=0}^{\ell} (C_k)^{j^k}.$$

The algorithms $PShare$ and $PVerify$ can be defined analogously to $Fshare$ and $FVerify$:

- $PShare(s, \ell, L) \rightarrow \{s_j\}_{j=1}^{L}, \{s'_j\}_{j=1}^{L}, \{C_k\}_{k=0}^{\ell}$,
- $PVerify(s_j, s'_j, \{C_k\}_{k=0}^{\ell}) \rightarrow b$ where $b \in \{0, 1\}$.

Protocol. We give the modified DKG protocol Π_{DKG} from GJKR-DKG in Figure A.2. The participating parties can drop out, as long as $\eta_D + \delta_D < 1/3$.

Correctness and security. We analyze the properties of Π_{DKG} in this section. We start by revisiting the correctness and security definitions of GJKR-DKG, and then discuss how our definition differs from theirs because of a weakening of the communication model. In GJKR-DKG, correctness has three folds:

1. All subsets of honest parties define the same unique secret key.
2. All honest parties have the same public key.
3. The secret key is uniformly random.

Security means that no information about the secret key can be learned by the adversary except for what is implied by the public key.

For Π_{DKG}, if the server is honest, then our communication model (§2.2.3) is equivalent to having a fully synchronous channel, hence in this case the correctness and security properties in the prior work hold. When the server is malicious, we show that Π_{DKG} satisfies the following correctness (C1, C2, C3, C4) and security (S).

C1. Each honest party either has no secret at the end or agrees on the same QUAL with other honest parties.

C2. The agreed QUAL sets defines a unique secret key.

C3. The secret key defined by QUAL is uniformly random.

C4. Each honest party, either has no public key, or outputs the same public key with other honest parties.

S. Malicious parties learns no information about the secret key except for what is implied by the public key.

Lemma A.2.5. *Let the participants in DKG be L parties and a server. If $\delta_D + \eta_D < 1/3$, then under the communication model defined in Section 2.2.3, protocol Π_{DKG} (Fig. A.2) has properties C1, C2, C3, C4 and S in the presence of a malicious adversary controlling the server and up to η_D fraction of the parties.*

Proof. Since $L = 3\ell+1$, and by $\delta_D+\eta_D < 1/3$, at most ℓ are malicious (the dropouts can be treated as malicious parties who do not send the prescribed messages). We first show under which cases the honest parties will have no share. Note that the parties that are excluded from $\mathcal{D}_2$ are those who are honest but did not receive a complaint against a malicious party who performed wrong sharing; and parties that are excluded from $\mathcal{D}_3$ are those who have complained at i in step (b) but did not receive shares from i at this step. If the server drops messages (sent from one online honest party to another) in the above two cases, then in the cross-check step, the honest parties will not receive more than $2\ell+1$ QUALs, and hence will abort. In this case, honest parties in the end has no share.

Now we prove C1 by contradiction. Suppose there are two honest parties P_1 and P_2 at the end of the protocol who holds secrets (not abort) and they have different QUAL. Then this means P_1 received at least $2\ell + 1$ same QUAL sets S_1 and P_2 received at least same $2\ell + 1$ QUAL sets S_2. W.l.o.g., assume that there are $\ell - v$ malicious parties ($v \geq 0$). In the $2\ell+1$ sets S_1, at least $\ell+1+v$ of them are from honest parties. Similarly, for the $2\ell + 1$ sets S_2, at least $\ell + 1 + v$ are from other honest parties different than above (since an honest party cannot send both S_1 and S_2). However, note that we have in total $2\ell + 1 + v$ honest parties, which derives a contradiction.

Recall that at most ℓ parties are malicious, so the QUAL set has at least $\ell+1$ parties, and since we have C1, now C2 is guaranteed. Moreover, since QUAL contains at least one honest party, the secret key is uniform (C3). C4 follows exactly from the work GJKR. The proof for S is the same as GKJR, except that the simulator additionally simulates the agreement protocol in Lemma A.4.1. □

Remark 17. An important difference between Π_{DKG} and standard DKG protocols is that we allow aborts and allow honest parties to not have shares at the end of the protocol. When some honest parties do not have shares of the secret key, the server is still able to get sum (decryption still works)

because malicious parties hold the shares.

Protocol Π_{DKG} based on discrete log

Parameters. A set of L parties (denoted as $\mathcal{D}_0$), threshold ℓ where $3\ell + 1 = L$. $\delta_D + \eta_D < 1/3$.

Protocol outputs. A subset of the L parties hold secret sharing of a secret key $s \in \mathbb{Z}_q$; the server holds the public key g^s signed by all the clients.

Notes. The parties have access to PKI. All messages sent from one party to another via the server are signed and end-to-end encrypted.

1. Each party i performs verifiable secret sharing (VSS) as a dealer:
   (a) *Share*:

   Party $i \in \mathcal{D}_0$ randomly chooses $s_i \in \mathbb{Z}_q$, computes $\{s_{i,j}\}_{j=1}^{L}, \{s'_{i,j}\}_{j=1}^{L}, \{C_{i,k}\}_{k=0}^{\ell} \leftarrow PShare(s_i, \ell, L)$.

   It also computes $\{A_k\}_{k=0}^{\ell}$ from $FShare(s, \ell, L)$ and stores it locally.

   Send $s_{i,j}$ and $s'_{i,j}$ to each party j, and $\{C_{i,k}\}_{i=0}^{\ell}$ to all parties $j \in \mathcal{D}_0$ via the server.

   // Denote the set of parties who received all the prescribed messages after this step as $\mathcal{D}_1$.

   (b) *Verify and complain*:

   Each party $j \in \mathcal{D}_1$ checks whether it received at least $(1 - \delta_D)L$ valid signed shares. If not, abort; otherwise continue.

   Each party $j \in \mathcal{D}_1$, for each received share $s_{i,j}$, runs $b \leftarrow PVerify(j, s_{i,j}, s'_{i,j}, \{C_{i,k}\}_{k=0}^{\ell})$.

   If b is 1, then party i does nothing; otherwise party i sends (`complaints`, j) to all the parties in $\mathcal{D}_0$ via the server.

   // Denote the set of parties who received all the prescribed messages after this step as $\mathcal{D}_2$.

   (c) *Against complaint*:

   Each party $i \in \mathcal{D}_2$, who as a dealer, if received a valid signed (`complaint`, i) from j, sends $s_{i,j}, s'_{i,j}$ to all parties in $\mathcal{D}_0$ via the server.

   // Denote the set of parties who received all the prescribed messages after this step as $\mathcal{D}_3$.

   (d) *Disqualify*:

   Each party $i \in \mathcal{D}_3$ marks any party j as disqualified if it received more than $2\ell + 1$ valid signed (`complaints`, j), or party j answers with $s_{j,i}, s'_{j,i}$ such that $PVerify(s_{j,i}, s'_{j,i}, \{C_{j,k}\}_{k=1}^{\ell})$ outputs 0. The non-disqualified parties form a set QUAL.

   Each party $i \in \mathcal{D}_3$ signs the QUAL set and sends to all parties in $\mathcal{D}_0$ to the server. The server, on receiving a valid signed QUAL, signs and sends it to all parties in $\mathcal{D}_3$.

   (e) *Cross-check QUAL*:

   Each party $i \in \mathcal{D}_3$ checks whether it receives at least $2\ell + 1$ valid signed QUAL, if so, they sum up the shares in QUAL and derive a share of secret key. If not, abort.

Figure A.2: Protocol Π_{DKG} (Part I).

Protocol Π_{DKG} (Cont'd) based on discrete log

1. Compute public key:
   (a) Each party $i \in$ QUAL sends $\{A_{i,k}\}_{k=1}^{\ell}$ to all parties via the server.
   (b) Each party i runs $b' \leftarrow FVerify(s_{j,i}, \{A_{j,k}\}_{k=1}^{\ell})$ for $j \in$ QUAL. If b' is 0, then party i sends to all the parties in $\mathcal{D}_3 \cap$ QUAL via the server a message (`complaint`, $j, s_{j,i}, s'_{j,i}$) for those $(s_{j,i}, s'_{j,i})$ such that b is 1 and b' is 0.

      // For $b = 1$ and $b' = 0$: The check in step 1.d) passes but fails this step

   (c) For each j such that (`complaint`, $j, s_{j,i}, s'_{j,i}$) is valid, parties reconstruct s_j. For all parties in QUAL, set $y_i = g^{s_i}$, and compute $PK = \prod_{i \in QUAL} y_i$. Parties in QUAL sign PK using their own signing keys and send the signed PK to the server.

Figure A.3: Protocol Π_{DKG} (Part II).

A.2.3 Collection phase

The detailed protocol for each round in the collection phase is described in Figure A.4. At the beginning of round t, the server notifies the clients who should be involved, namely S_t. A client who gets a notification can download public keys of its neighbors $A_t(i)$ from PKI server (the server should tell clients how to map client IDs to the names in PKI). To reduce the overall run time, clients can pre-fetch public keys used in the coming rounds.

A.2.4 Transfer shares

Every R rounds, the current set of decryptors $\mathcal{D}$ transfer shares of SK to a new set of decryptors, $\mathcal{D}_{new}$. To do so, each $u \in \mathcal{D}$ computes a destination decryptors set $\mathcal{D}_{new}$ for round t, by ChooseSet$(v, \lceil t/R \rceil, L, N)$. Assume now each decryptor $u \in \mathcal{D}$ holds share s_u of SK (i.e., there is a polynomial p such that $p(u) = s_u$ and $p(0) = SK$). To transfer its share, each $u \in \mathcal{D}$ acts as a VSS dealer exactly the same as the first part in Π_{DKG} (Fig.2.9) to share s_u to new decryptor $j \in \mathcal{D}_{new}$. In detail, u chooses a polynomial p^*_u of degree ℓ and sets $p^*_u(0) = s_u$ and all other coefficients of p^*_u to be random. Then, u sends $p^*_u(j)$ to each new decryptor $j \in \mathcal{D}_{new}$.

Each new decryptor $j \in \mathcal{D}_{new}$ receives the evaluation of the polynomials at point j (i.e., $p^*_u(j)$ for all $u \in \mathcal{D}$). The new share of SK held by j, s'_j, is defined to be a linear combination of the received shares: $s'_j := \sum_{u \in \mathcal{D}} \beta_u \cdot p^*_u(j)$, where the combination coefficients $\{\beta_u\}_{u \in \mathcal{D}}$ are constants (given the

Collection phase: Π_{sum} for round t out of T total rounds

Initial state from setup phase: each client $i \in [N]$ holds a value v and public key $PK = g^s$ where $SK = s$; each decryptor $u \in \mathcal{D}$ additionally holds a Shamir share of SK (threshold ℓ with $3\ell+1 = L$). We require $\delta_D + \eta_D < 1/3$. Steps for round t:

1. **Report step.**

   Server performs the following:

   Compute a set $\mathcal{Q}_{graph} \leftarrow \textsc{ChooseSet}(v, t, n_t, N)$ and a graph $G_t \leftarrow \textsc{GenGraph}(v, t, \mathcal{Q}_{graph})$; store $\{A_t(i)\}_{i \in \mathcal{Q}_{graph}}$ computed from G_t.

   Notify each client $i \in \mathcal{Q}_{graph}$ that collection round t begins.

   Each client $i \in \mathcal{Q}_{graph}$ performs the following:

   Compute $\mathcal{Q}^{local}_{graph} \leftarrow \textsc{ChooseSet}(v, t, n_t, N)$, and if $i \notin \mathcal{Q}^{local}_{graph}$, ignore this round.

   Read from PKI g^{a_j} for $j \in A_t(i)$, and compute $r_{i,j}$ by computing $(g^{a_j})^{a_i}$ and mapping it to $\{0,1\}^\lambda$.

   Sample $m_{i,t} \xleftarrow{\$} \{0,1\}^\lambda$ and compute $\{h_{i,j,t}\}_{j \in A(i)} \leftarrow \text{PRF}(r_{ij}, t)$ for $j \in A_t(i)$.

   Send to server a message $msg_{i,t}$ consisting of

   - $Vec_{i,t} = \mathbf{x}_{i,t} + \text{PRG}(m_{i,t}) + \sum_{j \in A_t(i)} \pm\text{PRG}(h_{i,j,t})$,
   - $\text{SymAuthEnc}(k_{i,u}, m_{i,u,t} \| t)$, for $u \in \mathcal{D}$,
   - $\text{AsymEnc}(PK, h_{i,j,t})$ for $j \in A_t(i)$

   where $m_{i,u,t} \leftarrow \mathit{Share}(m_{i,t}, \ell, L)$, $A_t(i) \leftarrow \textsc{FindNeighbors}(v, S_t, i)$, and AsymEnc (ElGamal) and SymAuthEnc (authenticated encryption) are defined in Appendix A.2.1; along with the signatures $\sigma_{i,j,t} \leftarrow \mathit{Sign}(sk_i, c_{i,j,t} \| t)$ for all ciphertext $c_{i,j,t} = \text{AsymEnc}(PK, h_{i,j,t})$ $\forall j \in A_t(i)$.

2. **Cross check step.**

   Server performs the following:

   Denote the set of clients that respond within timeout as $\mathcal{Q}_{vec}$.

   Compute partial sum $\tilde{z}_t = \sum_{i \in \mathcal{Q}_{vec}} Vec_{i,t}$.

   Build decryption request req (req consists of clients in S_t to be labeled):

   Initialize an empty set $\mathcal{E}_i$ for each $i \in \mathcal{Q}_{graph}$, and

   if $i \in \mathcal{Q}_{vec}$, label i with "online",

   and add $\text{SymAuthEnc}(k_{i,u}, m_{i,u,t} \| t)$ to $\mathcal{E}_i$, where $k_{i,j}$ is derived from PKI (Appendix A.2.1);

   else label i with "offline",

   and add $\{(\text{AsymEnc}(PK, h_{i,j,t}), \sigma_{i,j,t}\}_{j \in A_t(i) \cap \mathcal{Q}_{vec})}$ to $\mathcal{E}_i$.

   Send to each $u \in \mathcal{D}$ the request req and $\mathcal{E}_i$ of all clients $i \in \mathcal{Q}_{graph}$.

   Each decryptor $u \in \mathcal{D}$ performs the following:

   Upon receiving a request req, compute $\sigma^*_u \leftarrow \mathit{Sign}(sk_u, req \| t)$, and send (req, σ^*_u) to all other decryptors via the server.

Figure A.4: Collection protocol Π_{sum} (Part I).

Collection phase (Cont'd): Π_{sum} for round t out of T total rounds

Initial state from setup phase: each client $i \in [N]$ holds a value v and public key $PK = g^s$ where $SK = s$; each decryptor $u \in \mathcal{D}$ additionally holds a Shamir share of SK (threshold ℓ with $3\ell+1 = L$). We require $\delta_D + \eta_D < 1/3$. Steps for round t:

1. **Reconstruction step.**

   Each decryptor $u \in \mathcal{D}$ performs the following:

   Ignore messages with signatures ($\sigma_{i,j,t}$ or σ^*_u) with round number other than t.

   Upon receiving a message $(req, \sigma^*_{u'})$, run $b \leftarrow VerSig(pk_i, req, \sigma^*_{u'})$. Ignore the message if $b = 0$.

   Continue only if u received $2\ell+1$ or more same req messages that were not ignored. Denote such message as req^*.

   For req^*, continue only if

   each client $i \in S_t$ is either labeled as "online" or "offline";

   the number of "online" clients is at least $(1-\delta)n_t$;

   all the "online" clients are connected in the graph;

   each online client i has at least k online neighbors such that $\eta^k < 2^{-\kappa}$.

   For each $i \in \mathcal{Q}_{graph}$,

   For each SymAuthEnc$(k_{i,u}, m_{i,u,t} \| t)$ in $\mathcal{E}_i$, use $k_{i,u}$ (derived from PKI) to decrypt; send $m_{i,u,t}$ to the server if the decryption succeeds;

   For each $(\text{AsymEnc}(PK, h_{i,j,t}), \sigma_{i,j,t}) \in \mathcal{E}_i$, parse as $((c_0, c_1), \sigma)$ and send $c_0^{s_u}$ to the server if $VerSig((c_0, c_1), \sigma)$ outputs 1;

   Server completes the sum:

   Denote the set of decryptors whose messages have been received as U. Compute a set of interpolation coefficients $\{\beta_u\}_{u \in U}$ from U.

   For each $i \in \mathcal{Q}_{graph}$, reconstruct the mask $m_{i,t}$ or $\{h_{i,j,t}\}_{j \in A_t(i) \cap \mathcal{Q}_{vec}}$:

   For each parsed (c_0, c_1) meant for $h_{i,j,t}$ in $\mathcal{E}_i$, compute $h_{i,j,t}$ as $c_1 \cdot (\prod_{u \in U} (c_0^{s_u})^{\beta_u})^{-1}$;

   For each set of received shares $\{m_{i,u,t}\}_{u \in U}$, compute $m_{i,t}$ as $Recon(\{m_{i,u,t}\}_{u \in U})$.

   Output $z_t = \tilde{z}_t - \text{PRG}(m_{i,t}) + \sum_{j \in A_t(i) \cap \mathcal{Q}_{vec}} \pm\text{PRG}(h_{i,j,t})$.

Figure A.5: Collection protocol Π_{sum} (Part II).

set $\mathcal{D}$, we can compute $\{\beta_u\}_{u\in\mathcal{D}}$). Note that the same issue about communication model for DKG also exists here, but the same relaxation applies.

Here we require that $\eta_D + \delta_D < 1/3$ for both $\mathcal{D}$ and $\mathcal{D}_{new}$. As a result, each client j in a subset $\mathcal{D} \subseteq \mathcal{D}_{new}$ holds a share $s_{u,j}$. For each receiving decryptor $j \in \mathcal{D}$, it computes $s'_j = \sum_{u\in\mathcal{D}_{old}} \beta_u \cdot s_{u,j}$, where each β_u is some fixed constant.

Since the sharing part is exactly the same as Π_{DKG} and the share combination happens locally, the same correctness and security argument of DKG applies. Specifically, for correctness, each honest party either has no secret at the end or agrees on the same QUAL with other honest parties; and the QUAL defines the unique same secret key before resharing. For security, a malicious adversary will not learn any information, but can cause the protocol aborts.

A.3 Requirements on Parameters

A.3.1 The number of decryptors

In this section, we show how to choose L such that, given N, η, δ_D, we can guarantee less than $1/3$ of the L chosen decryptors are malicious. Note that δ_D is given because this can be controlled by the server, i.e., the server can decide how long to wait for the decryptors to respond. On the other hand, η_D is a random variable.

To guarantee $2\delta_D + \eta_D < 1/3$ (Theorem 2.2.4), a necessary condition is that $\eta < 1/3 - 2\delta_D$. This can be formalized as a probability question: given N clients where η fraction of them are malicious, randomly sample L clients (decryptors) from them; the number of malicious clients X in the decryptors should follow the tail bound of hypergeometric distribution [BBG$^+$20, Section 2]:

$$\Pr[X \geq (\eta + (1/3 - 2\delta_D - \eta))L] \leq e^{-2L(1/3-\eta-2\delta_D)^2},$$

For η and δ_D both being 1%, the choice of $L = 60$ (which we used for benchmarks in §2.2.10) gives 1.6×10^{-5} probability. If we double L to be 120, then this guarantees 2.6×10^{-10} probability.

A.3.2 Proof of Lemma 2.2.2

The algorithm in Figure 2.1 gives a random graph $G(n, \epsilon)$. A known result in random graphs is, when the edge probability $\epsilon > \frac{(1+\omega)\ln n}{n}$, where ω is an arbitrary positive value, the graph is almost surely connected when n is large. Note that in BBGLR, they also use this observation to build the graph that has significant less number of neighbors than a prior work by Bonawitz et al. [BIK+17], but in their work since the clients chooses their neighbors, the resulting graph is a biased one; and they guarantee that there is no small isolated components in the graph.

Concretely, from Gilbert [Gil59], let $g(n, \epsilon)$ be the probability that graph $G(n, \epsilon)$ has disconnected components, and it can be recursively computed as

$$g(n, \epsilon) = 1 - \sum_{i=1}^{n-1} \binom{n-1}{i-1} g(i, \epsilon)(1-\epsilon)^{i(n-i)}.$$

Number of nodes n	128	512	1024
Parameter ϵ (failure probability 10^{-6})	0.11	0.03	0.02
Parameter ϵ (failure probability 10^{-12})	0.25	0.06	0.03

Figure A.6: Parameters ϵ to ensure random graph connectivity.

We numerically depict the above upper bound of the probability $g(n, \epsilon)$ for different ϵ in Figure A.6. For example, when $n = 1024$, to ensure less than 10^{-6} failure probability, we need $\epsilon \geq 0.02$, hence the number of neighbors a client needs when $\delta = \eta = 0.01$ is at least $\lceil (\epsilon + \delta + \eta)n \rceil = 41$.

A.4 Security proofs

A.4.1 Security definition

We say *a protocol* Π *securely realizes ideal functionality* $\mathcal{F}$ *in the presence of a malicious adversary* $\mathcal{A}$ if there exists a probabilistic polynomial time algorithm, or simulator, $\mathcal{S}$, such that for any probabilistic polynomial time $\mathcal{A}$, the distribution of the output of the real-world execution of Π is (statistically or computationally) indistinguishable from the distribution of the output of the ideal-world execution invoking $\mathcal{F}$: the output of both worlds' execution includes the inputs and outputs

Functionality $\mathcal{F}_{\text{sum}}^{(t)}$

Parties: clients in S_t and a server.

Parameters: dropout rate δ and malicious rate η over n_t clients.

- $\mathcal{F}_{\text{sum}}^{(t)}$ receives a set $\mathcal{O}_t$ such that $|\mathcal{O}_t|/|S_t| \leq \delta$, and from the adversary $\mathcal{A}$ a set of corrupted parties, $\mathcal{C}$; and $\mathbf{x}_{i,t}$ from client $i \in S_t \backslash (\mathcal{O}_t \cup \mathcal{C})$.
- $\mathcal{F}_{\text{mal}}$ sends S_t and $\mathcal{O}_t$ to $\mathcal{A}$, and asks $\mathcal{A}$ for a set M_t: if $\mathcal{A}$ replies with $M_t \subseteq S_t \backslash \mathcal{O}_t$ such that $|M_t|/|S_t| \geq 1-\delta$, then $\mathcal{F}_{\text{mal}}$ computes $\mathbf{z}_t = \sum_{i \in M_t \backslash \mathcal{C}} \mathbf{x}_{i,t}$ and continues; otherwise $\mathcal{F}_{\text{mal}}$ sends `abort` to all the honest parties.
- Depending on whether the server is corrupted by $\mathcal{A}$:
  - If the server is corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{mal}}$ outputs $\mathbf{z}_t$ to all the parties corrupted by $\mathcal{A}$.
  - If the server is not corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{mal}}$ asks $\mathcal{A}$ for a shift $\mathbf{a}_t$ and outputs $\mathbf{z}_t + \mathbf{a}_t$ to the server.

Figure A.7: Ideal functionality for round t in collection phase.

of honest parties the view of the adversary $\mathcal{A}$. In our proof, we focus on the computational notion of security. Note that $\mathcal{S}$ in the ideal world has one-time access to $\mathcal{F}$, and has the inputs of the corrupted parties controlled by $\mathcal{A}$.

A.4.2 Ideal functionalities

We provide ideal functionality for Flamingo in Figure 2.4. Looking ahead in the proof, we also need to define an ideal functionality for the setup phase and an ideal functionality for a single round in the collection phase, which we give as Figure 2.3 and Figure A.7.

We model the trusted source of initial randomness as a functionality $\mathcal{F}_{\text{rand}}$; that is, a party or ideal functionality on calling $\mathcal{F}_{\text{rand}}$ will receive a uniform random value v in $\{0,1\}^\lambda$, where λ is the length of the PRG seed.

A.4.3 Proof of Theorem 2.2.1

The ideal functionality $\mathcal{F}_{\text{setup}}$ for the setup phase is defined in Figure 2.3. Depending on whether the server is corrupted or not, we have the following two cases.

1. When the server is not corrupted, then the communication model is equivalent to a secure broadcast channel. By security of GJKR, Π_{setup} securely realizes $\mathcal{F}_{\text{setup}}$.

2. When the server is corrupted, we build a simulator for the adversary $\mathcal{A}$. We start by listing the messages that $\mathcal{A}$ sees throughout the setup phase:

   - A random value v from $\mathcal{F}_{\text{rand}}$;
   - All the messages in Π_{DKG} that are sent via the server;
   - All the messages in Π_{DKG} that are seen by the corrupted decryptors.

   The simulator first calls $\mathcal{F}_{\text{rand}}$ and receives a value v. Then the simulator interacts with $\mathcal{A}$ acting as the honest decryptors. The simulator aborts if any honest decryptors would abort in Π_{DKG}. There are two ways that $\mathcal{A}$ can cheat: 1) $\mathcal{A}$ cheats in Π_{DKG}, and in Appendix A.2.2, we show the simulator can simulate the view of $\mathcal{A}$; 2) $\mathcal{A}$ cheats outside Π_{DKG}, this means $\mathcal{A}$ chooses a wrong set of decryptors, or it broadcasts wrong signatures. So the simulator aborts as long as it does not receive $2\ell + 1$ valid signatures on PKs signed by the set defined by v.

A.4.4 Proof of Theorem 2.2.3

The proof for dropout resilience is rather simple: in the setup phase, at most δ_D fraction of L selected decryptors drop out; then in one round of the collection phase, another δ_D fraction of decryptors can drop out. Since $2\delta_D + \eta_D < 1/3$, and $3\ell + 1 = L$ (Fig. A.2), the online decryptors can always help the server to reconstruct the secrets.

A.4.5 Proof of Theorem 2.2.4

We first present the proof for a single round: the collection protocol Π_{sum} (Fig. A.4) for round t securely realizes the ideal functionality $\mathcal{F}_{\text{sum}}^{(t)}$ (Fig. A.7) in the random oracle model. From the ideal functionality $\mathcal{F}_{\text{sum}}^{(t)}$ we can see that the output sum is not determined by the actual dropout set $\mathcal{O}_t$, but instead M_t sent by the adversary.

In the proof below for a single round, for simplicity, we omit the round number t when there is no ambiguity. We assume the adversary $\mathcal{A}$ controls a set of clients in $[N]$, with the constraint $2\delta_D + \eta_D < 1/3$. Denote the set of corrupted clients in $[N]$ as $\mathcal{C}$ and as before the set of the decryptors is $\mathcal{D}$; the malicious decryptors form a set $\mathcal{C} \cap \mathcal{D}$ and $|\mathcal{C} \cap \mathcal{D}| < L/3$. From the analysis

in Appendix A.1, we have $|\mathcal{C}|/|S_t| \leq \eta$.

Case 1. We start with the case where the server is corrupted by $\mathcal{A}$. Now we construct a simulator $\mathcal{S}$ in the ideal world that runs $\mathcal{A}$ as a subroutine. We assume both the simulator $\mathcal{S}$ and ideal functionality $\mathcal{F}_{\text{sum}}^{(t)}$ (Fig. A.7) have access to an oracle $\mathcal{R}_{\text{drop}}$ that provides the dropout sets $\mathcal{O}_t$. In other words, the dropout set is not provided ahead of the protocol but instead provided during the execution (similar notion appeared in prior work [BIK+17]). Assume that in the ideal world, initially a secret key SK is shared among at least $2L/3$ clients in $\mathcal{D}$. The simulation for round t is as follows.

1. $\mathcal{S}$ received a set $\mathcal{O}_t$ from the oracle $\mathcal{R}_{\text{drop}}$.

2. $\mathcal{S}$ receives a set M_t from the adversary $\mathcal{A}$.

3. $\mathcal{S}$ obtains $\mathbf{z}_t$ from $\mathcal{F}_{\text{sum}}^{(t)}$.

4. (Report step) $\mathcal{S}$ interacts with $\mathcal{A}$ as in the report step acting as the honest clients $i \in M_t \backslash \mathcal{C}$ with masked inputs $\mathbf{x}'_i$, such that $\sum_{i \in M_t \backslash \mathcal{C}} \mathbf{x}'_i = \mathbf{z}_t$.

   Here the input vector $\mathbf{x}'_i$ and the mask m_i are generated by $\mathcal{S}$ itself, and the pairwise secrets are obtained by querying the PKI.

5. (Cross-check step) $\mathcal{S}$ interacts with $\mathcal{A}$ acting as honest decryptors as in the cross-check step.

6. (Reconstruction step) $\mathcal{S}$ interacts with $\mathcal{A}$ acting as honest decryptors in the reconstruction step, where $\mathcal{S}$ uses the shares of the secret key SK to perform decryption of honest parties.

7. In the above steps, if all the honest decryptors would abort in the protocol prescription then $\mathcal{S}$ sends `abort` to $\mathcal{F}_{\text{sum}}^{(t)}$, outputs whatever $\mathcal{A}$ outputs and halts.

We construct a series of hybrid execution, starting from the real world to the ideal world execution.

Hybrid 1. The view of $\mathcal{A}$ in the real world execution is the same as the view of $\mathcal{A}$ in the ideal world when $\mathcal{S}$ would have the actual inputs of honest parties, $\{\mathbf{x}_i\}_{i \in S_t \backslash (\mathcal{C} \cup \mathcal{O}_t)}$, the pairwise and individual

masks, and the shares of the secret key SK. ($\mathcal{S}$ in fact would know SK in full because $3\ell + 1 = L$ and that the number of honest parties is $2\ell + 1$ or more.)

Hybrid 2. $\mathcal{S}$ now instead of using the actual secret key s, it replaces s with 0 and sends the corresponding $|\mathcal{C} \cap \mathcal{D}| < L/3$ shares of 0 in $\mathbb{Z}_q$ to $\mathcal{A}$. The joint distribution of less than $L/3$ shares (recall that the threshold is ℓ where $3\ell + 1 = L$), from the property of Shamir secret sharing, for s and 0 are the same. Hence this hybrid has identical distribution to the previous hybrid.

Hybrid 3. $\mathcal{S}$ now instead of using the actual pairwise masks between honest parties, it samples a random pairwise mask $r'_{i,j}$ from $\{0,1\}^\lambda$ and computes the corresponding ElGamal ciphertext as (c'_0, c'_1). $\mathcal{S}$ does not change the pairwise mask between a client controlled by $\mathcal{A}$ and an honest client (such pairwise mask can be obtained by querying PKI to get g^{a_j} for an honest client j, and compute $(g^{a_j})^{a_i}$ for malicious client i). We argue that $\mathcal{A}$'s view in this hybrid is computationally indistinguishable from the previous one as below.

First, we need to assume the mapping from $\mathbb{G}$ to $\{0,1\}^\lambda$ is a random oracle. To specify, in the real world, the mask $r_{i,j}$ is computed from the mapping on $g^{a_i a_j}$; and in the ideal world the mask $r'_{i,j}$ is randomly sampled. Let M_t be the set of online clients that $\mathcal{A}$ labels in the real world (recall the server is controlled by $\mathcal{A}$). $\mathcal{A}$ in both worlds observes $\mathrm{PRF}(r_{i,j}, t)$ between a client i out of M_t and a client j in M_t, hence we require $r_{i,j}$ to be random as a PRF key.

Second, $\mathcal{A}$ in the ideal world does not observe the pairwise masks between clients in M_t, but only the ciphertexts generated from $r'_{i,j}$ for those clients; and the distribution of the ciphertexts is computationally indistinguishable from what $\mathcal{A}$ observed from the real world by the security of ElGamal encryption (Definition A.2.2).

Hybrid 4. $\mathcal{S}$ now instead of using symmetric encryption (SymAuthEnc) of the shares of the actual individual mask m_i, it uses the symmetric encryption of a randomly sampled m'_i from $\{0,1\}^\lambda$ as the individual mask. Looking ahead in the proof, we also need to model the PRG as a random oracle $\mathcal{R}_{\mathrm{PRG}}$ that can be thought of as a "perfect PRG" (see more details in a prior work [BIK+17]). For

all $i \in M_t/\mathcal{C}$, $\mathcal{S}$ samples Vec_i at random and programs $\mathcal{R}_{\text{PRG}}$ to set $\text{PRG}(m'_i)$ as

$$\text{PRG}(m'_i) = Vec_i - \mathbf{x}_i - \sum \pm \text{PRG}(r'_{i,j}),$$

where the vectors Vec_i's are vectors observed in the real world. The view of $\mathcal{A}$ regarding Vec_i's in this hybrid is statistically indistinguishable to that in the previous hybrid.

Moreover, $\mathcal{A}$ learns the m_i in the clear for those $i \in M_t$ in both worlds, and the distributions of those m_i's in the ideal and real world are identical; for those m_i's where $i \notin M_t$ that $\mathcal{A}$ should not learn, from the semantic security of the symmetric encryption scheme (Definition A.2.3), and the threshold $\ell < L/3$, $\mathcal{A}$'s view in this hybrid is computationally indistinguishable from the previous one.

Hybrid 5. $\mathcal{S}$ now instead of programming the oracle $\mathcal{R}_{\text{PRG}}$ as in the previous hybrid, it programs the oracle as

$$\text{PRG}(m'_i) = Vec_i - \mathbf{x}'_i - \sum \pm \text{PRG}(r'_{i,j}),$$

where $\mathbf{x}'_i$'s are chosen such that $\sum_{i \in M_t \setminus \mathcal{C}} \mathbf{x}_i = \sum_{i \in M_t \setminus \mathcal{C}} \mathbf{x}'_i$. From Lemma 6.1 in a prior work [BIK$^+$17] under the same setting, the distribution of $\mathcal{A}$'s view in this hybrid is statistically indistinguishable to that in the previous hybrid, except probability $2^{-\kappa}$: if the graph becomes disconnected or there is an isolated node after removing $\mathcal{O}_t$ and $\mathcal{C}$ from S_t, then the server learns x_i in the clear and thus can distinguish between the two worlds. When $\mathcal{A}$ cheats by submitting M_t to $\mathcal{S}$ where the graph formed by nodes in M_t is not connected, $\mathcal{S}$ simulates honest decryptors and output abort. In this case, the distribution of $\mathcal{A}'s$ view in the ideal world is the same as that in the real world.

Hybrid 7. Same as the previous hybrid, except that the label messages req from honest decryptors in the last step are replaced with the offline/online labels obtained from the oracle. In all steps, $\mathcal{A}$ would cheat by sending invalid signatures to $\mathcal{S}$; in this case $\mathcal{S}$ will abort. In the cross-check and reconstruction steps, there are following ways that $\mathcal{A}$ would cheat here:

1. $\mathcal{A}$ sends multiple different M_t's to $\mathcal{F}_{\text{sum}}$. $\mathcal{S}$ in the ideal world will simulate the protocol in Lemma A.4.1 below, and outputs whatever the protocol outputs.

2. $\mathcal{A}$ sends to $\mathcal{F}_{\text{sum}}$ a set M_t with less than $(1-\delta)n_t$ clients, or the clients in M_t are disconnected, or there is a client in M_t with less than η^k online neighbors. In this case, $\mathcal{S}$ will abort, which is the same as in the real-world execution.

The last hybrid is exactly the ideal world execution. To better analyze the simulation succeeding probability, we use κ_1 to denote the security parameter for the graph connectivity (Lemma 2.2.2) and use κ_2 to denote the security parameter for the third checking in the cross-check round (§2.2.7). The simulation can fail in two ways: 1) The graph gets disconnected (even when the server is honest); 2) There exists a client in S_t such that all of its online neighbors are malicious. The former happens with probability $2^{-\kappa_1}$. The latter is bounded by $n \cdot 2^{-\kappa_2}$: the probability that the opposite event of 2) happens is $(1-\eta^k)^n \approx 1 - n\eta^k$ (assuming η^k is very small). Thus the failure probability $n\eta^k \leq n \cdot 2^{-\kappa_2}$.

Case 2. For the case where the server is not corrupted by $\mathcal{A}$, the simulation is the same as Case 1, except that the simulator needs to compose the "shifts" added by $\mathcal{A}$ in each step to hit the value $\mathbf{a}_t$.

This completes the proof that for any single round $t \in [T]$, the protocol Π_{sum} for round t securely realizes $\mathcal{F}_{\text{sum}}^{(t)}$ when $\delta_D + \eta_D < 1/3$, except probability $2^{-\kappa_1} + n \cdot 2^{-\kappa_2} \leq n \cdot 2^{-\kappa+1}$, where $\kappa = \min\{\kappa_1, \kappa_2\}$.

Lemma A.4.1. *Assume there exists a PKI and a secure signature scheme; there are $3\ell+1$ parties with at most ℓ colluding malicious parties. Each party has an input bit of 0 or 1 from a server. Then there exists a one-round protocol for honest parties to decide if the server sent the same value to all the honest parties.*

Proof. If an honest party receives $2\ell+1$ or more messages with the same value, then it means the server sends to all honest parties the same value. If an honest party receives less than $2\ell+1$ messages with the same value, it will abort; in this case the server must have sent different messages to different honest parties. □

Remark 18. The above analysis of the agreement protocol shows where the threshold 1/3 comes from. Consider the case where the threshold is $1/2$ and $2\ell+1 = L$. For a target client, the (malicious) server can tell $\ell/2$ decryptors that the client is online and tell another $\ell/2+1$ decryptors that the client is offline. Then combined with the ℓ malicious decryptors' shares, the server has $\ell/2+\ell$ shares to reconstruct individual mask, and $\ell/2+1+\ell$ shares to reeconstruct the pairwise mask.

Multi-round security. Our threat model assumes that $\mathcal{A}$ controls ηN clients throughout T rounds (§2.2.3). There are two things we need to additionally consider on top of the single-round proof: 1) the set S_t is generated from PRG, and 2) the pairwise mask $h_{i,j,t}$ computed from $\mathrm{PRF}(r_{i,j}, t)$. For the former, we program $\mathcal{R}_{\mathrm{PRG}}$ (like the single-round proof) such that the ChooseSet outputs S_t.

Now we analyze the per-round pairwise masks. Let the distribution of the view of $\mathcal{A}$ in round t be Δ_t. We next show that if there exists an adversary $\mathcal{B}$, and two round number $t_1, t_2 \in [T]$ such that $\mathcal{B}$ can distinguish between Δ_{t_1} and Δ_{t_2}, then we can construct an adversary $\mathcal{B}'$ who can break PRF security. We call the challenger in PRF security game simply as challenger. There are two worlds (specified by $b = 0$ or 1) for the PRF game. When $b = 0$, the challenger uses a random function; when $b = 1$, the challenger uses PRF and a random key for the PRF. We construct $\mathcal{B}'$ as follows. On input t_1, t_2 from $\mathcal{B}$, $\mathcal{B}'$ asks challenger for h_{i,j,t_1} for all clients i and j, and round t_1, t_2. Then $\mathcal{B}'$ creates the messages computed from $h_{i,j,t}$'s as protocol Π_{sum} prescribed; it generates two views $\Delta_{t_1}, \Delta_{t_2}$ and sends to $\mathcal{B}$. $\mathcal{B}'$ outputs whatever $\mathcal{B}$ outputs.

Failure probability for T rounds. For a single round, we already showed that protocol $\Pi^{(t)}_{\mathrm{sum}}$ securely realizes $\mathcal{F}^{(t)}_{\mathrm{sum}}$ except probability $p = n \cdot 2^{-\kappa+1}$. The probability that for all the T rounds the protocol is secure is therefore $1-(1-p)^T$, which is approximately $1 - T \cdot p$ when $T \cdot p \ll 1$. Therefore, the probability of failure (there exists a round that fails the simulation) is $Tn2^{-\kappa+1}$.

APPENDIX B

DEFERRED MATERIALS FOR ARMADILLO PROTOCOL

B.1 Approximate proof of smallness

We describe the approximate proof of smallness from Lyubashevsky et al. [LNS21]. Let σ be a security parameter for this approximate proof. The prover has a vector $\mathbf{w}$ of length m where $\|\mathbf{w}\|_\infty < B$. Let B' be the bound that the prover can prove with the following protocol and the gap $\gamma := B'/B$ should be larger than $19.5\sigma\sqrt{m}$.

1. The prover first sends $\mathsf{com}(\mathbf{w})$ to the verifier.
2. The prover chooses a uniform length-σ vector $\mathbf{y} \leftarrow [\pm\lceil b/2(1+1/\sigma)\rceil]^\sigma$, and sends $\mathsf{com}(\mathbf{y})$ to the verifier.
3. The verifier chooses $\mathbf{R} \leftarrow \mathcal{D}^{\sigma\times m}$ and sends it to the prover, where each entry in $\mathbf{R}$ being zero with probability 1/2, being 1 with probability 1/4 and -1 with probability 1/4.
4. The prover computes $\mathbf{u} := \mathbf{R}\cdot\mathbf{w}$ and $\mathbf{z} = \mathbf{u} + \mathbf{y}$. It restarts the protocol from Step 2 if either $\|\mathbf{u}\|_\infty > b/2\lambda$ or $\|\mathbf{z}\| > b/2$.
5. The prover sends $\mathbf{z}$ to the verifier.
6. The verifier chooses a random r and sends r to the prover.
7. The prover and the verifier run an inner-product proof that

$$\langle \mathbf{R}^\top\mathbf{r}, \mathbf{w}\rangle + \langle \mathbf{r}, \mathbf{y}\rangle = \langle \mathbf{R}^\top\mathbf{r}|\mathbf{r}, \mathbf{w}|\mathbf{y}\rangle = \langle \mathbf{z}, \mathbf{r}\rangle,$$

   where $\mathbf{r} = (r^0, r^1, \ldots, r^{\sigma-1})$.

Note that $\langle \mathbf{z}, \mathbf{r}\rangle$ and $\mathbf{R}^\top\mathbf{r}|\mathbf{r}$ are known to both the prover and the verifier. The last step is essentially a length-$(m+\sigma)$ linear proof.

Secure aggregation for training iteration t

Server and clients agree on public parameters:

- LWE parameters $(\lambda, \ell, p, q, \mathbf{A} \in \mathbb{Z}_q^{\ell \times \lambda})$ and $\Delta = \lfloor q/p \rfloor$.
- Proof parameters: Let $\mathbb{G}$ be the group of order q for the commit-and-proof system. Let $\mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{K}$ be vectors of generators in $\mathbb{G}$ of length λ, ℓ, ℓ, C. The norm bounds are $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}}(L_\infty)$.
- System model parameters: dropout rate is δ and malicious rate over this iteration of selected clients is η.

Setup. The set $\mathcal{D}$ of helpers is determined independently from the aggregation as described in Section 2.3.4, with threshold being d. Let the secret key and public key for $j \in \mathcal{D}$ be (SK_j, PK_j). Let $k_{i,j}$ be MAC key shared between client i and helper $j \in \mathcal{D}$; such key can be derived from PKI.

Round 1 (Server → Clients)

Server notifies a set S_t of n clients (indexed by numbers in $[n]$) to start iteration $t \in [T]$. It also tells the helpers the IDs of the n clients. Each helper $j \in \mathcal{D}$ derives the MAC key $k_{i,j}$ for each $i \in S_t$.

Round 1 (Clients → Server)

Client $i \in S_t$ on input $\mathbf{x}_i \in \mathbb{Z}_q^m$, computes the following:

1. Compute $\mathbf{y}_i = \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i \bmod q$, where $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda$, $\mathbf{e}_i \leftarrow \chi^m$. // For outer aggregation
2. Compute degree-d packed secret sharing of $\mathbf{s}_i$ as $\boldsymbol{\rho}_i = (\rho_i^{(1)}, \ldots, \rho_i^{(D)})$. // For inner aggregation
3. Compute commitments $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i), \mathsf{com}_{\mathbf{G}}(\mathbf{e}_i), \mathsf{com}_{\mathbf{H}}(\mathbf{x}_i)$; and $\mathsf{com}_{K_j}(\rho_i^{(j)})$ for $j \in \mathcal{D}$, where $\mathbf{K}$ is parsed as $\{K_j\}_{j \in \mathcal{D}}$.
4. Set constraint system $\mathbb{CS}_{\text{shares}}$:

$$\{\mathsf{io} : (\mathsf{com}(\boldsymbol{\rho}_i), \mathsf{com}(\mathbf{s}_i), \mathbf{w}), \quad \mathsf{st} : \langle \boldsymbol{\rho}_i | \mathbf{s}_i, \mathbf{w} \rangle = 0, \quad \mathsf{wt} : (\boldsymbol{\rho}_i, \mathbf{s}_i)\},$$

   and compute $\pi_{\text{shares}} \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, where $m^*(X) \leftarrow_{\$} \mathbb{F}[X]_{\leq D+\lambda-d-2}$ and $\mathbf{w} := (v_1 \cdot m^*(1), \ldots, v_n \cdot m^*(D))$.
5. Set constraint system $\mathbb{CS}_{\text{enc}}$:

$$\begin{aligned}&\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i)),\\ &\mathsf{st} : \mathbf{y}_i = \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i, \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2), \|\mathbf{e}\|_\infty < B_{\mathbf{e}}(L_\infty), \|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{s}_i, \mathbf{x}_i, \mathbf{e}_i)\},\end{aligned}$$

   and compute $\pi_{\text{enc}} \leftarrow \Pi_{\text{enc}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$.
6. Send a tuple to the server:

$$\begin{aligned}&\{\text{“server”} : (\mathbf{y}_i, \mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i), \pi_{\text{shares}}, \pi_{\text{enc}});\\ &\text{“helper } j \in \mathcal{D}\text{”} : \mathsf{ct}_j := \text{AsymEnc}(PK_j, \rho_i^{(j)}), \mathsf{com}_{K_j}(\rho_i^{(j)}) \text{ and a MAC tag } \sigma_{i,j} \leftarrow \mathsf{Mac}(k_{i,j}, \mathsf{ct}_{i,j})\}\end{aligned}$$

Figure B.1: Armadillo protocol description for computing a single sum privately (Part I).

Secure aggregation for training iteration t contd.

Round 2 (Server → Helpers)

Let $\mathcal{X}_1$ be the clients who sent the prescribed messages in Round 1.

The server for each client $i \in \mathcal{X}_1$ computes:

1. Compute $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) := \prod_{j\in[C]} \mathsf{com}_{K_j}(\rho_i^{(j)})$.
2. Run $\Pi_{\text{linear}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{shares}})$ and $\Pi_{\text{enc}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{enc}})$.
3. Remove all clients with invalid proof from $\mathcal{X}_1$. Call this set $\mathcal{X}_2$.
4. If all the proofs are valid, forward messages intended for $j \in \mathcal{D}$.

Round 2 (Helpers → Server)

Each helper $j \in \mathcal{D}$: for every i,

1. Check if $\sigma_{i,j}$ is valid. If there are fewer than $(1-\delta-\eta)n$ valid messages, abort. Otherwise continue.
2. It computes $\rho_i^{(j)} := \text{AsymDec}(SK_j, \mathsf{ct}_j)$, and checks if it is consistent with $\mathsf{com}_{K_j}(\rho_i^{(j)})$. If not, create a verifiable complaint that consists of $\rho_i^{(j)}$ and the proof of decryption of ct_j; denote this proof as π_{dec}.
3. It formed a set $\mathcal{V}_j$ that consists of all the clients whose shares are valid.

Round 3 (Server → Helpers)

Server tells all the helpers a set of clients who were complained about, denoted as $\mathcal{B}$. Set $\mathcal{S}_3 := \mathcal{S}_2 \setminus \mathcal{B}$.

Round 3 (Helpers → Server)

Each helper $j \in \mathcal{D}$:

1. Remove clients in $\mathcal{B}$ from $\mathcal{V}_j$.
2. Compute $\rho^{(j)} := \sum_{i\in\mathcal{V}_j} \rho_i^{(j)}$ and send it to the server.

Server reconstructs the shares $\{\rho^{(j)}\}_{j\in\mathcal{D}}$ to $\mathbf{s}$, and computes $\mathbf{y} := \sum_{i\in\mathcal{X}_2\setminus\mathcal{B}} \mathbf{y}_i$ and computes $\lfloor \mathbf{y} - \mathbf{A}\cdot\mathbf{s} \mod q \rceil_\Delta$.

Figure B.2: Armadillo protocol description for computing a single sum privately (Part II).

B.2 Proof of decryption

We will instantiate the public key encryption using RSA cryptosystem. We choose two large primes p, q and let $N = pq$, and we choose an integer e such that e is coprime to $p-1$ and $q-1$. The public key is (N, e) and the private key is (N, d) where $d = e^{-1} \bmod (p-1)(q-1)$. An integer $0 \le m < N$ is encrypted as $c = m^e \bmod N$ and the decryption is computed as $m = c^d \bmod N$. Then proof of decryption relative to public key (N, e) is simple: a decryptor reveals the decryption result m (claimed to be inconsistent with the commitment) to the server, and the server checks if m^e equals c.

B.3 Full protocol for proof of encryption

We now describe proof protocol Π_{enc}, which is proof of Regev's encryption with bounded norms for error and input.

Recall the constraints that client i wishes to prove are

$$\begin{aligned}\mathbb{CS}_{\text{enc}} : \{&\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i)),\\ &\mathsf{st} : \mathbf{y}_i = \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta\mathbf{x}_i,\\ &\qquad \|\mathbf{x}_i\|_2 < B_{\mathbf{x}}(L_2),\\ &\qquad \|\mathbf{x}_i\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{e}_i\|_\infty < B_{\mathbf{e}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{s}_i, \mathbf{x}_i, \mathbf{e}_i)\}\end{aligned}$$

This can be proven using the techniques we presented in Section 2.3.4 and 2.3.4. Below we omit index i for simplicity. Assuming the prover already commits to $\mathbf{x}, \mathbf{e}, \mathbf{s}$.

1. The prover computes $\mathbf{y} = \mathbf{A}\mathbf{s} + \mathbf{e} + \Delta\mathbf{x}$.
2. The prover chooses random r from $\mathbb{Z}_q$ and let $\mathbf{r} := (r^0, \ldots, r^{\ell-1})$, and invokes an inner product proof on $\langle \mathbf{y}, \mathbf{r} \rangle = \langle \mathbf{A}^\top \mathbf{r} | \mathbf{r} | \Delta\mathbf{r}, \mathbf{s} | \mathbf{e} | \mathbf{x} \rangle$.
3. The prover invokes proof of L_2 norm (described in §2.3.4) on $\mathbf{x}$.
4. The prover invokes proof of L_∞ norm (described in §2.3.4) on $\mathbf{e}$ and $\mathbf{x}$.

B.4 Security proof

We give our full protocol description in Figures B.1 and B.2. Below we give proof of Theorem 3.5.4.

Our proof methodology relies on the standard simulation-based proof, where we show that every adversary attacking our protocol can be simulated by an adversary Sim in an ideal world where the functionality $\mathcal{F}$ (Fig.2.11). In the following, we first prove privacy against any adversary corrupting ηn clients and the server; then we prove robustness assuming the adversary corrupting ηn clients but not the server (recall our threat model in §2.2.3).

The challenge in the simulation is the ability of Sim to generate a valid distribution for the honest clients' inputs, even without knowing their keys. To this end, we will show that Sim, when only given the sum of the user inputs $\mathbf{X} = \sum_{i=1}^{n} \mathbf{x}_i$, can simulate the expected leakage for the server which includes n ciphertexts, the sum of the n keys $\mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i$, and such that the sum of the n ciphertexts, when decrypted with $\mathbf{K}$, correctly decrypts to $\mathbf{X}$.

Before we detail the definition of Sim and prove its security, we present an assumption that we will use later.

Definition B.4.1 (A variant of Hint-LWE [LKK+18, CKK+21])**.** Consider integers λ, m, q and a probability distribution χ' on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the Hint-LWE assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{m\times\lambda}, \mathbf{k} \leftarrow\!\!\$\ \mathbb{Z}_q^{\lambda}, \mathbf{e} \leftarrow\!\!\$\ \chi'^{m} \\ \mathbf{r} \leftarrow\!\!\$\ \mathbb{Z}_q^{\lambda}, \mathbf{f} \leftarrow\!\!\$\ \chi'^{m} \\ \mathbf{y}_0 := \mathbf{A}\mathbf{k} + \mathbf{e}, \mathbf{y}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^{m}, b \leftarrow\!\!\$\ \{0,1\} \\ b' \leftarrow\!\!\$\ \mathcal{A}(\mathbf{A}, (\mathbf{y}_b, \mathbf{k}+\mathbf{r}, \mathbf{e}+\mathbf{f})) \end{array} \right] = \frac{1}{2} + \mathsf{negl}(\kappa)$$

where κ is the security parameter.

Intuitively, Hint-LWE assumption says that $\mathbf{y}_0$ looks pseudorandom to an adversary, even when given some randomized leakage on the secret and the error vectors. Kim et al. [KLSS23] show that solving Hint-LWE is no easier than solving LWE problem. For a secure LWE instance (λ, m, q, χ) where χ is a discrete Gaussian distribution with standard deviation σ, the corresponding Hint-LWE instance (λ, m, q, χ'), where χ' is a discrete Gaussian distribution with standard deviation σ', is secure when $\sigma' = \sigma/\sqrt{2}$. Consequently, any $\mathbf{e} \in \chi$ can be written as $\mathbf{e}_1 + \mathbf{e}_2$ where $\mathbf{e}_1, \mathbf{e}_2 \in \chi'$. This

gives us the real distribution $\mathcal{D}_R$, with the error term re-written and the last ciphertext modified.

$$\left\{ \begin{array}{c|c} \mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i \bmod q & \forall i \in [n], \mathbf{k}_i \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \leftarrow\!\!\$\ \chi'^m \\ \mathbf{y}_1, \ldots, \mathbf{y}_n & \forall i \in [n-1], \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + \mathbf{e}_i + \Delta \mathbf{x}_i \\ & \mathbf{y}_n = \mathbf{AK} - \sum_{i=1}^{n-1} \mathbf{y}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

We now define $\mathsf{Sim}(\mathbf{A}, \mathbf{X})$:

$\underline{\mathsf{Sim}(\mathbf{A}, \mathbf{X})}$

Sample $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1} \leftarrow\!\!\$\ \mathbb{Z}_q^m$

Sample $\mathbf{k}_1, \ldots, \mathbf{k}_n \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda$

Sample $\mathbf{e}_1, \ldots, \mathbf{e}_n \leftarrow\!\!\$\ \chi'^m$

Sample $\mathbf{f}_1, \ldots, \mathbf{f}_n \leftarrow\!\!\$\ \chi'^m$

Set $\mathbf{K} := \sum_{i=1}^{n} \mathbf{k}_i \bmod q$

Set $\mathbf{u}_n = \mathbf{A} \cdot \mathbf{K} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \cdot \mathbf{X}$

Return $\mathbf{K}, \mathbf{u}_1, \ldots, \mathbf{u}_n$

In other words, the simulated distribution, $\mathcal{D}_{\mathsf{Sim}}$, is:

$$\left\{ \begin{array}{c|c} \mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i \bmod q & \forall\ i \in [n]\ \mathbf{k}_i \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \leftarrow\!\!\$\ \chi'^m \\ \mathbf{u}_1, \ldots, \mathbf{u}_n & \forall\ i \in [n-1]\ \mathbf{u}_i \leftarrow\!\!\$\ \mathbb{Z}_q^m \\ & \mathbf{u}_n = \mathbf{AK} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

We will now prove that $\mathcal{D}_R$ is indistinguishable from $\mathcal{D}_{\mathsf{Sim}}$ through a sequence of hybrids.

- Hybrid 0: This is $\mathcal{D}_R$.
- Hybrid 1: In this hybrid, we will replace the real ciphertext $\mathbf{y}_1$ with a modified one. In other

words, we set:

$$\left\{\begin{array}{c|l} \mathbf{K} & \forall\ i \in [n]\ \mathbf{k}_i \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \leftarrow\!\!\$\ \chi'^m, \mathbf{u}'_1 \leftarrow\!\!\$\ \mathbb{Z}_q^m \\ \mathbf{y}_1 = \mathbf{u}'_1 + \mathbf{f}_1 + \Delta\mathbf{x}_1 & \forall\ i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A}\cdot\mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{AK} - \sum_{i=1}^{n-1}\mathbf{y}_i + \sum_{i=1}^{n}(\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{X} \end{array}\right\}$$

Now, we will show that if there exists an adversary $\mathcal{B}$ that can distinguish between Hybrid 0 and 1, then we can define an adversary $\mathcal{A}$ who can distinguish the two ensembles in the Hint-LWE Assumption. Let us define $\mathcal{A}$ now.

$\underline{\mathcal{A}(\mathbf{A}, \mathbf{y}^*, \mathbf{k}^* = \mathbf{k} + \mathbf{r} \bmod q, \mathbf{e}^* = \mathbf{e} + \mathbf{f})}$

Sample $\mathbf{k}_2, \ldots, \mathbf{k}_{n-1} \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda$

Sample $\mathbf{e}_2, \ldots, \mathbf{e}_n \leftarrow\!\!\$\ \chi'^m$

Sample $\mathbf{f}_2, \ldots, \mathbf{f}_n \leftarrow\!\!\$\ \chi'^m$

Set $\mathbf{K} = \sum_{i=2}^{n-1}\mathbf{k}_i + \mathbf{k}^* \bmod q$ // implicitly, $\mathbf{k}_n := \mathbf{r}$

$\forall\ i \in \{2, \ldots, n-1\},\ \mathbf{y}_i = \mathbf{A}\mathbf{k}_i + \mathbf{e}_i + \mathbf{f}_i + \Delta\mathbf{x}_i$

Set $\mathbf{y}_1 = \mathbf{y}^* + \mathbf{f}_n + \Delta\mathbf{x}_1$

Set $\mathbf{y}_n := \mathbf{AK} - \sum_{i=1}^{n-1}\mathbf{y}_i + \mathbf{e}^* + \sum_{i=2}^{n}(\mathbf{e}_i + \mathbf{f}_i) + \Delta\cdot\mathbf{X}$

Run $b' \leftarrow\!\!\$\ \mathcal{B}(\mathbf{K}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$

return b'

We need to argue that the reduction correctly simulates the two hybrids, based on the choice of y^*.

- If $\mathbf{y}^* = \mathbf{Ak}+\mathbf{e}$, then $\mathbf{y}_1$ is a valid encryption of $\mathbf{x}_1$ with key $\mathbf{k}$ and error $(\mathbf{e}+\mathbf{f}_n)$. Further, it is easy to verify that $\mathbf{y}_n$ satisfies the definition present in Hybrid 0.
- If $\mathbf{y}^* = \mathbf{u}$ for some random $\mathbf{u}$. Then, we get that $\mathbf{y}_n$ is of the prescribed format, while also guaranteeing that $\mathbf{y}_1$ is generated as expected.

- Hybrid 2: In this hybrid, we will replace $\mathbf{y}_1$ with $\mathbf{y}_1$ that is sampled uniformly at random.

$$\left\{\begin{array}{c|c} \mathbf{K} & \forall\ i \in [n]\ \mathbf{k}_i \leftarrow\!\!\$\ \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \leftarrow\!\!\$\ \chi'^m, \mathbf{u}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^m \\ \mathbf{u}_1 & \forall\ i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{AK} - \mathbf{u}_1 - \sum_{i=2}^{n-1} \mathbf{y}_i + \sum_{i=1}^{n} (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array}\right\}$$

Hybrid 1, and Hybrid 2 are identically distributed $\mathbf{u}_1'$ is uniformly sampled and essentially mask the values in $\mathbf{y}_1$ of Hybrid 1.

In Hybrids 3 and 4, we replace $\mathbf{y}_2$ with a random element $\mathbf{u}_2$, by using a similar logic. Therefore, in Hybrid $2n-2$, the distribution will resemble $\mathcal{D}_{\mathsf{Sim}}$. This concludes the proof of simulatability.

Privacy against a semi-honest server. Here we prove privacy against an attacker corrupting the server and a set of ηn clients (some of them can be helpers). Denote the simulator as Sim_p. Here, the server acts semi-honestly. The formal proof proceeds through a sequence of hybrids. The sequence of hybrids is similar to the work of Bell et al. [BBG+20]. Let $\mathcal{H} = [n] \setminus \mathcal{C}$. Below, we detail the hybrids.

- Hybrid 0: This is the real execution of the protocol where the adversary is interacting with honest parties.

- Hybrid 1: This is where we introduce a simulator Sim which knows all the inputs and secret keys involved, i.e., it knows the keys and the shares of all the clients. Sim runs a full execution of the protocol with the adversary and programs the random oracle as needed. The view of the adversary in this hybrid is indistinguishable from the previous hybrid.

- Hybrid 2: Our next step is for the simulator Sim to rely on the Special Honest Verifier Zero Knowledge (SHVZK) property of all the proof systems to simulate the zero-knowledge proofs for each honest client. Any non-negligible distinguishing advantage between Hybrids 1 and 2 will violate the SHVZK property of the underlying proof systems.

- Hybrid 3: In the next step, we rely on the hiding property of Pedersen commitments. Recall

that the hiding property guarantees that there is a negligible distinguishing advantage for an adversary between an actual Pedersen commitment and a random group element. Therefore, for all the honest clients, Sim can simply replace the commitments provided with a random group element. Any non-negligible distinguishing advantage between Hybrids 2 and 3 will violate the hiding property of the commitment scheme.

- Hybrid 4: In the next step, we rely on the privacy property of Shamir Secret Sharing. This guarantees that any insufficient number of shares does not leak the privacy of the secret. In this hybrid Sim uses this property to replace the shares of the honest user's keys meant for the corrupt helpers with random values. Recall that the number of corrupt helpers is strictly less than the reconstruction threshold. Therefore, any non-negligible advantage in distinguishing advantage between Hybrids 3 and 4 will imply that the statistical security of Shamir's Secret Sharing is broken.

  Thus far, for the honest clients' Sim has successfully generated all the contributions for the honest users, except for the ciphertexts themselves. However, Sim cannot simply rely on the semantic security of LWE encryption to replace with encryptions of random values. This is because the output might differ from the real world. Instead, Sim, which has control of the corrupted parties, simply instructs the corrupted parties to provide their inputs as $\mathbf{0}$. Then, the output of the functionality is simply the sum of the honest clients' inputs. Let us call it $\mathbf{x}_H$. With this knowledge, Sim can generate its own choices of individual inputs for honest clients, with the only constraint that the values necessarily need to sum up $\mathbf{x}_H$. This guarantees that the output is correct.

- Hybrid 5: Sim now relies on the semantic security of LWE encryption, under leakage resilience as argued earlier in this section, to instead encrypt these sampled values for honest clients. Any non-negligible distinguishing advantage between Hybrids 4 and 5 will imply that the LWE encryption is no longer semantically secure.

At Hybrid 5, it is clear that Sim can successfully simulate a valid distribution that does not rely on

the honest party's inputs. This concludes the proof.

Robustness. Now we turn to proving robustness (and also showing privacy) when the adversary corrupts only a set of ηn clients (some can be helpers). Here, the server follows the protocol but can try to violate the privacy.

We denote the simulator here as Sim_r. Note that in the ideal world, Sim_r has to provide the inputs for both the honest and corrupted clients. Meanwhile, in the real world, the inputs for the corrupted clients come from the adversary, call it $\mathcal{B}$. Note that $\mathcal{B}$ can choose these inputs with any restrictions. Therefore, to ensure that it produces a valid set of inputs to the functionality in the ideal world, Sim_r does the following:

- It invokes $\mathcal{B}$ by internally running it. Sim_r honestly follows the protocol, fixing the inputs for the honest clients to be some valid vector $\mathbf{X}$. To $\mathcal{B}$, this is an expected run, and therefore, it behaves exactly like in the real-world execution.

- Sim_r records the set of corrupted parties $\mathcal{A}$ and the set of dropout clients $\mathcal{O}$ encountered in this internal execution.

- At some point, $\mathcal{B}$ provides the NIZK proofs to the server for adversarial clients. However, Sim_r controls the server with these proofs including proof of Shamir sharing, proof of correct encryption, range proofs, and the proof of binding of shares and the key.

- Using the Knowledge Soundness property of the NIZK proofs, Sim_r is able to extract the witnesses, specifically the inputs for the adversarial clients.

- Finally, Sim_r also records whatever $\mathcal{B}$ outputs in the internal execution.

With these steps in place, Sim_r can simulate the ideal world.

- It sends the recorded $\mathcal{O}, \mathcal{A}$ to the ideal functionality.

- It sends the extracted adversarial inputs for those clients, while sending the valid inputs for

the non-dropout honest clients.

- Note that the inputs in both the real-world and ideal-world match. We need to show that the computed output matches too.

- Finally, Sim_r outputs whatever $\mathcal{B}$ had output in the internal execution.

It is clear that the output of Sim_r (in the ideal world) is indistinguishable from the output of $\mathcal{B}$ (in the real world). However, we now need to argue that the output sum cannot differ at all. Specifically, while it is guaranteed that the adversarial inputs are included in the sum in the real world (as it was done in the internal execution of $\mathcal{B}$). We need to show that the honest clients' inputs cannot be dropped from the computed sum.

To see this, observe that the server only removes a client if there is a proof of the client misbehaving. As a corollary, it implies that an honest party's input is never rejected by the honest server as it would not have proof of malicious behavior. This guarantees that any honest client's inputs, which hasn't dropped out, is always included in the computed sum in the real world. In other words, the computed sum in the real and ideal world have to match.

APPENDIX C

DEFERRED PROOF OF PIR CHAPTER

C.1 Deferred materials for impossibility results

Attack for linear PIR. We say that a PIR is *linear* if its encoding function is linear; that is, for any two databases x and x', $P_{x+x'} = P_x + P'_x$. Most multi-server PIR schemes (essentially linear smooth locally decodable codes) considered in existing literature are linear.

Theorem C.1.1 (Attacks for linear PIR)**.** *Any linear PIR in the shuffle model, when the total number of queries C is less than the database size n, has statistical security no better than $\frac{n-C}{n-1}$.*

At a high level, the attacker simply checks whether or not the value at a given index is determined by the linear constraints imposed by the observed queries. Upon choosing a suitable basis for the domain and range, a linear encoding function can be represented by a generating matrix, which we denote as $\mathbf{M}$. Let M_q denote the row vector corresponding to query q. We will show that when the number of queries is less than n, we can narrow down the set of possible client queries by at least 1.

Lemma C.1.2. *Let $e_i \in \mathbb{F}^n$ denote the i-th standard basis vector. Let Q_i^k denote the support of* $\mathsf{Query}(i; n)$*. For every $(q_1, \ldots, q_k) \in Q_i^k$, we have $e_i \in \operatorname{span}\{M_{q_1}, \ldots M_{q_k}\}$.*

Proof of Lemma C.1.2. Assume that e_i is not in the span corresponding to $\mathbf{q} = (q_1, \ldots, q_k) \in Q_i^k$. Intuitively, this should mean that the i-th entry of the database cannot be fully determined by these queries. We formalize this intuition in showing that there must exist a database on which Recon fails.

First, we set some notation. Let V denote the vector space spanned by $M_{q_1}, \ldots M_{q_k}$. Let $\mathbf{M_q}$ be the matrix formed by the subrows of $\mathbf{M}$ corresponding to the queries $\mathbf{q}$.

Now we show that there must exist a vector w in the null space of $\mathbf{M_q}$ such that $w_i \neq 0$. We can prove this by contradiction. Assume that for every $w \in \operatorname{null}(\mathbf{M_q})$, $w_i = 0$. In other words, $\operatorname{null}(\mathbf{M_q})$

is orthogonal to e_i. Let $\mathbf{M_q}'$ be $\mathbf{M_q}$ with e_i as an additional row. The previous observation ensures that $\text{null}(\mathbf{M_q}) = \text{null}(\mathbf{M_q}')$. By rank nullity, $\text{rank}(\mathbf{M_q}) = \text{rank}(\mathbf{M_q}')$. We conclude that e_i does not add to the rank of $\mathbf{M_q}$; therefore, e_i can be written as a linear combination of $M_{q_1}, \ldots, M_{q_k}$. This contradicts our assumption that e_i is not in the span of these vectors.

Finally, we show that Recon will fail with some positive probability. Let $x \in \mathbb{F}^n$ be some arbitrary database. Let w be a vector in the null space of $\mathbf{M_q}$ such that $w_i \neq 0$. Then $x_i \neq (x+w)_i$. When given the answers to these particular queries $q_1, \ldots, q_k$, Recon cannot distinguish between x and $x + w$, yet they have different values at index i. Therefore, Recon must fail for at least one of x or $x + w$. □

Proof of Theorem C.1.1. Applying the above lemma when the total number of queries is $C < n$, we know that the span of the corresponding row vectors of M will be of dimension at most C, so there will be at least $n - C$ standard basis vectors $e_{i_1}, \ldots e_{i_{n-C}}$ missing from the span. $i_1, \ldots, i_{n-C}$ must not have been in the original set of client indices.

This leads to a natural candidate for a distinguisher. We will show an adversary which has advantage at least $\frac{n-C}{n-1}$ in distinguishing between all-0 vector $(0, \ldots, 0)$ and all-i vector $(i, \ldots, i)$, when the total number of queries is $C < n$. i is some particular index which will depend on the specific PIR.

The distinguisher. If $e_i \notin \text{span}(M_{q_1}, \ldots, M_{q_k})$, then output 0; else output 1.

Claim. *There exists $i \in [n]$ such that the above distinguisher has advantage $\frac{1}{n-1}$ between the all-0 vector and the all-i vector.*

To see why this claim holds, notice that there are the following two cases:

- Case 0 (all-0 vector): in each realization of the queries, there is at least $n - C$ indices whose basis vectors are not in the span of the queries. Since there are $n - 1$ of these i's, by linearity of expectation, there must exist some i with probability at least $\frac{n-C}{n-1}$ of being excluded from the span. For the said i, the probability the distinguisher outputs 0 is $\frac{n-C}{n-1}$.

- Case 1 (all-i vector): by Lemma C.1.2, e_i is in the span of each of the sharings of i. Therefore, it is in the span of the aggregate of all the shares. The distinguisher outputs 0 with probability 0.

We conclude that the difference in probabilities of the two cases is $\frac{n-C}{n-1}$. □

C.2 Imperfect shuffling

Definition C.2.1 (Imperfect shuffler)**.** A shuffler $\Pi = \{\Pi_c\}_{c\in\mathbb{N}}$ where each Π_c is a distribution over the symmetric group $\mathfrak{S}_c$ is said to be at most ζ-imperfect if for all c,

$$\max_{X\sim\Pi_c} \Pr[X = \sigma] \leq \zeta \cdot \Pr_{Y\sim\widetilde{\Pi}_c}[Y = \sigma]$$

where $\widetilde{\Pi} = \{\widetilde{\Pi}_c\}_{c\in\mathbb{N}}$ is the uniform shuffler. In other words, a ζ-imperfect allows for the probability of any particular shuffling to be at most a factor of ζ larger than the uniform case.

We now illustrate the robustness of the constructions to imperfect shuffling. In particular, if Π' is ζ-imperfect for a constant ζ, then the statistical distance of our construction when Π' is used is at most ζ times the statistical distance when $\widetilde{\Pi}$ is used. The following lemma shows this more generically:

Lemma C.2.2. *Consider any distributions $\mathcal{D}_\Pi$ and $\mathcal{D}'_\Pi$ that depend on a shuffler Π on group $\mathbb{G}$. Let $\widetilde{\Pi}$ be the uniform shuffler on group $\mathbb{G}$ and Π' be a ζ-imperfect shuffler. Then, $\mathsf{SD}(\mathcal{D}_{\Pi'}, \mathcal{D}'_{\Pi'}) \leq \zeta \cdot \mathsf{SD}(\mathcal{D}_{\widetilde{\Pi}}, \mathcal{D}'_{\widetilde{\Pi}})$.*

Proof.

$$\begin{aligned}
\mathsf{SD}(\mathcal{D}_{\Pi'}, \mathcal{D}'_{\Pi'}) &= \sum_{\sigma} \Pr[\Pi' = \sigma] \cdot \mathsf{SD}((\mathcal{D}_{|\sigma}, \mathcal{D}'_{|\sigma})) \\
&\leq \sum_{\sigma} \zeta \cdot \Pr[\widetilde{\Pi} = \sigma] \cdot \mathsf{SD}((\mathcal{D}_{|\sigma}, \mathcal{D}'_{|\sigma})) \\
&= \zeta \sum_{\sigma} \Pr[\widetilde{\Pi} = \sigma] \cdot \mathsf{SD}((\mathcal{D}_{|\sigma}, \mathcal{D}'_{|\sigma})) \\
&= \zeta \cdot \mathsf{SD}(\mathcal{D}_{\widetilde{\Pi}}, \mathcal{D}'_{\widetilde{\Pi}})
\end{aligned}$$

□

C.3 Complete security proof for Add-ShPIR (Theorem 3.5.1)

We now provide the full details of the security proof for Add-ShPIR—our generic composition that uses any k-server PIR as OPIR and 2-additive PIR as IPIR. Recall that our proof outline consists of three major steps; the subsequent subsections formally describe each of these steps.

C.3.1 Bounding the OPIR edit distance (proof of Lemma 3.5.2)

We start with the details for our first major proof step, namely bounding the edit distance between the OPIR sub-queries. This only requires proving Lemma 3.5.2, which we do below.

Proof. When balls are thrown according to the distribution $\mathcal{B}$, define $\mathbb{U}_\alpha$ to be the random variable for the number of balls thrown into bin α, and $\mathbb{U}_{b,\alpha}$ to be the indicator variable that is 1 exactly when the b^{th} ball is thrown into bin α and 0 otherwise.

Note that $\mathbb{U}_{b,\alpha}$ and $\mathbb{U}_{b',\alpha}$ are independent when $b \neq b'$ the balls are thrown in a pairwise independent fashion. Now, each $\mathbb{U}_{b,\alpha}$ is a Bernoulli random variable with parameter $1/N$. Therefore, $\mathbb{E}[\mathbb{U}_{b,\alpha}] = \frac{1}{N}$ and $\mathrm{Var}[\mathbb{U}_{b,\alpha}] = \frac{1}{N}\left(1 - \frac{1}{N}\right)$.

Now, by linearity of expectation, for all bins α, we have $\mathbb{E}[\mathcal{U}_\alpha] = B/N$. Furthermore, since the balls are thrown in a pairwise independent way, the variance is also linear, and therefore, $\mathrm{Var}[\mathbb{U}_\alpha] =$

$\sum_{b=1}^{B} \mathrm{Var}[\mathbb{U}_{b,\alpha}] = \frac{B}{N} \cdot \left(1 - \frac{1}{N}\right)$. Therefore,

$$\mathbb{E}[\mathbb{U}_\alpha^2] = \mathrm{Var}[\mathbb{U}_\alpha] + (\mathbb{E}[\mathbb{U}_\alpha])^2 = \frac{BN - B + B^2}{N^2}.$$

Similarly define $\mathbb{V}_\alpha$ and $\mathbb{V}_{b,\alpha}$ when the balls are thrown according to distribution $\mathcal{B}'$. Note that all the above analysis also carries over for $\mathbb{V}_\alpha$. Now,

$$\begin{aligned}
\mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|] &\leq \sqrt{\mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|^2]} = \sqrt{\mathbb{E}[\mathbb{U}_\alpha^2 + 2\mathbb{U}_\alpha\mathbb{V}_\alpha + \mathbb{V}_\alpha^2]} \\
&= \sqrt{\mathbb{E}[\mathbb{U}_\alpha^2] + 2\mathbb{E}[\mathbb{U}_\alpha]\mathbb{E}[\mathbb{V}_\alpha] + \mathbb{E}[\mathbb{V}_\alpha^2]} \\
&= \sqrt{\frac{2BN - 2B}{N^2}} \leq \sqrt{\frac{2B}{N}}.
\end{aligned}$$

where the first inequality is from the fact that $(\mathbb{E}[\mathbb{X}])^2 \leq \mathbb{E}[\mathbb{X}^2]$ for any random variable $\mathbb{X}$, and the third step is from linearity of expectation and the fact that $\mathbb{U}_\alpha$ and $\mathbb{V}_\alpha$ are independent.

Finally using the linearity of expectation again, we can compute the expected edit distance as:

$$\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}[\mathsf{ED}(\mathbf{u},\mathbf{v})] = \frac{1}{2}\mathbb{E}\left[\sum_\alpha |\mathbb{U}_\alpha - \mathbb{V}_\alpha|\right] \leq \frac{N}{2}\sqrt{\frac{2B}{N}} = \sqrt{\frac{BN}{2}}.$$

□

C.3.2 Bounding the edit distance of IPIR shares

We now provide details for our second major proof step on bounding the edit distance between the IPIR shares. We start by showing that when looking at the final statistical distance, it is enough to only consider parts that differ between $\mathbf{u}$ and $\mathbf{v}$. In particular, we prove the following statement.

Lemma C.3.1. *Consider $\ell \geq 0$ amd two (B, N)-valid configurations $\mathbf{u}$ and $\mathbf{v}$. Then,*

$$\mathsf{SD}(\mathsf{Share}^\ell_{\mathbf{u}}, \mathsf{Share}^\ell_{\mathbf{v}}) \leq \mathsf{SD}(\mathsf{Share}^\ell_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}^\ell_{\mathbf{v}\ominus\mathbf{u}})$$

Proof. Let $f_{\mathbf{u}}, f_{\mathbf{v}}, f_{\mathbf{u}\ominus\mathbf{v}}, f_{\mathbf{v}\ominus\mathbf{u}}$ denote the probability mass functions of $\mathsf{Share}^k_{\mathbf{u}}, \mathsf{Share}^\ell_{\mathbf{v}}, \mathsf{Share}^\ell_{\mathbf{u}\ominus\mathbf{v}}$ and

$\mathsf{Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}}$ respectively. Define $\mathbf{c} = \mathbf{u} \sqcap \mathbf{v}$ and let $f_{\mathbf{c}}$ be the probability mass function of $\mathsf{Share}^{\ell}_{\mathbf{c}}$. Now,

$$\begin{aligned}\mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}}, \mathsf{Share}^{\ell}_{\mathbf{v}}) &= \frac{1}{2}\sum_{\mathbf{w}} |f_{\mathbf{u}}(\mathbf{w}) - f_{\mathbf{v}}(\mathbf{w})| \\ &= \frac{1}{2}\sum_{\mathbf{w}} \left| \left(\sum_{\mathbf{w}' \leq \mathbf{w}} f_{\mathbf{c}}(\mathbf{w} \ominus \mathbf{w}') f_{\mathbf{u}\ominus\mathbf{v}}(\mathbf{w}') \right) - \left(\sum_{\mathbf{w}' \leq \mathbf{w}} f_{\mathbf{c}}(\mathbf{w} \ominus \mathbf{w}') f_{\mathbf{v}\ominus\mathbf{u}}(\mathbf{w}') \right) \right|\end{aligned}$$

by marginalization and since $\mathbf{w}'$ and $\mathbf{w} \ominus \mathbf{w}'$ deal with separate initial balls which would make their sharing independent. We now get,

$$\begin{aligned}\mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}}, \mathsf{Share}^{\ell}_{\mathbf{v}}) &= \frac{1}{2}\sum_{\mathbf{w}} \left| \sum_{\mathbf{w}' \leq \mathbf{w}} f_{\mathbf{c}}(\mathbf{w} \ominus \mathbf{w}') \left(f_{\mathbf{u}\ominus\mathbf{v}}(\mathbf{w}') - f_{\mathbf{v}\ominus\mathbf{u}}(\mathbf{w}') \right) \right| \\ &\leq \frac{1}{2}\sum_{\mathbf{w}} \sum_{\mathbf{w}' \leq \mathbf{w}} f_{\mathbf{c}}(\mathbf{w} \ominus \mathbf{w}') \left| f_{\mathbf{u}\ominus\mathbf{v}}(\mathbf{w}') - f_{\mathbf{v}\ominus\mathbf{u}}(\mathbf{w}') \right| \\ &= \frac{1}{2}\sum_{\mathbf{w}'} \left(\left| f_{\mathbf{u}\ominus\mathbf{v}}(\mathbf{w}') - f_{\mathbf{v}\ominus\mathbf{u}}(\mathbf{w}') \right| \sum_{\mathbf{w} \geq \mathbf{w}'} f_{\mathbf{c}}(\mathbf{w} \ominus \mathbf{w}') \right) \\ &\leq \frac{1}{2}\sum_{\mathbf{w}'} \left| f_{\mathbf{u}\ominus\mathbf{v}}(\mathbf{w}') - f_{\mathbf{v}\ominus\mathbf{u}}(\mathbf{w}') \right| \cdot 1 \\ &= \mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}})\end{aligned}$$

□

This allows us to restrict our attention to only $\mathbf{u} \ominus \mathbf{v}$ and $\mathbf{v} \ominus \mathbf{u}$ which are (δ, B)-valid configuration. We will now find the edit distance after splitting each of the δ balls into two additive shares. Formally, we show the following lemma:

Lemma C.3.2. *Consider two (δ, N)-valid configurations $\mathbf{u}$ and $\mathbf{v}$. Then,*

$$\mathbb{E}[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}}, \mathsf{Share}_{\mathbf{v}})] \leq \sqrt{2\delta N}.$$

Proof. When balls are thrown according to the distribution $\mathsf{Share}_{\mathbf{u}}$, define $\mathbb{U}_{\alpha}$ as the random variable

for the number of balls thrown into bin α. Define $\mathbb{V}_\alpha$ for distribution $\mathsf{Share}_{\mathbf{v}}$. First observe by linearity of expectation that:

$$\mathbb{E}[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}}, \mathsf{Share}_{\mathbf{v}})] = \frac{1}{2}\sum_{\alpha} \mathbb{E}\left[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|\right]$$

Now, to find the distribution of $\mathbb{U}_\alpha$, we need to find when additively splitting a ball results in an addition to the bin α. Let $(b_1, \ldots, b_\delta)$ denote the vector of balls in $\mathbf{u}$, and define $\mathbb{U}_{i,\alpha}$ to be the number of additive shares of ball b_i that go into bin α. Observe that for any particular α, all $\mathbb{U}_{i,\alpha}$ are independent and that $\mathbb{U}_\alpha = \sum_{i=1}^{\delta} \mathbb{U}_{i,\alpha}$. Now, consider two cases for each ball b_i, and a bin α:

1. $b_i = \alpha + \alpha$ (in the group $\mathbb{G}$). In this case, if the first additive share of b_i is sampled as α, then both additive shares will go into bin α; otherwise no share will go into bin α. This means that $\mathbb{U}_{i,\alpha} \sim 2 \cdot \mathsf{Ber}(1/N)$.

2. $b_i \neq \alpha + \alpha$ (in the group $\mathbb{G}$). In this case, if the first additive share of b_i is sampled either as α or $b_i - \alpha$, then exactly one of the additive shares will go into bin α; otherwise no share will go into bin α. This means that $\mathbb{U}_{i,\alpha} \sim \mathsf{Ber}(2/N)$.

Assume that there are $\lambda_{\mathbf{u},\alpha}$ balls that satisfy the first case and $\delta - \lambda_{\mathbf{u},\alpha}$ balls that satisfy the second case. Using the fact that the $\mathbb{U}_{i,\alpha}$ are independent, we can now compute the distribution of $\mathbb{U}_\alpha$ as:

$$\mathbb{U}_\alpha \sim 2 \cdot \mathsf{Binomial}(\lambda_{\mathbf{u},\alpha}, 1/N) + \mathsf{Binomial}(\delta - \lambda_{\mathbf{u},\alpha}, 2/N).$$

Consequently, the following hold:

$$\mathbb{E}[\mathbb{U}_\alpha] = \frac{2\lambda_{\mathbf{u},\alpha}}{N} + \frac{2(\delta - \lambda_{\mathbf{u},\alpha})}{N} = \frac{2\delta}{N}.$$

$$\mathrm{Var}[\mathbb{U}_\alpha] = 4\lambda_{\mathbf{u},\alpha}\frac{N-1}{N^2} + (\delta - \lambda_{\mathbf{u},\alpha})\frac{2(N-2)}{N^2} = \frac{2N\lambda_{\mathbf{u},\alpha} + 2N\delta - 4\delta}{N^2}.$$

Similarly, we can compute

$$\mathbb{E}[\mathbb{V}_\alpha] = \frac{2\delta}{N} \qquad \text{and} \qquad \text{Var}[\mathbb{V}_\alpha] = \frac{2N\lambda_{\mathbf{v},\alpha} + 2N\delta - 4\delta}{N^2}.$$

where $\lambda_{\mathbf{v},\alpha}$ is the number of balls in $\mathbf{v}$ that are equal to $\alpha + \alpha$ (in group $\mathbb{G}$). Now applying the Jensen's inequality $\mathbb{E}[Z] \leq \sqrt{\mathbb{E}[Z^2]}$ to the random variable $|\mathbb{U}_\alpha - \mathbb{V}_\alpha|$, we get:

$$\begin{aligned}
\mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|] &\leq \sqrt{\mathbb{E}[(\mathbb{U}_\alpha - \mathbb{V}_\alpha)^2]} = \sqrt{\mathbb{E}[(\mathbb{U}_\alpha)^2] + \mathbb{E}[(\mathbb{V}_\alpha)^2] - 2 \cdot \mathbb{E}[\mathbb{U}_\alpha] \cdot \mathbb{E}[\mathbb{U}_\alpha]} \\
&= \sqrt{\frac{2N\lambda_{\mathbf{u},\alpha} + 2N\delta - 4\delta + 4\delta^2}{N^2} + \frac{2N\lambda_{\mathbf{v},\alpha} + 2N\delta - 4\delta + 4\delta^2}{N^2} - \frac{8\delta^2}{N^2}} \\
&= \sqrt{\frac{2\lambda_{\mathbf{u},\alpha} + 2\lambda_{\mathbf{v},\alpha} + 4\delta}{N} - \frac{8\delta}{N^2}} \\
&\leq \sqrt{\frac{2\lambda_{\mathbf{u},\alpha} + 2\lambda_{\mathbf{v},\alpha} + 4\delta}{N}} \\
&\leq \sqrt{\frac{8\delta}{N}}
\end{aligned}$$

since $0 \leq \lambda_{\mathbf{u},\alpha}, \lambda_{\mathbf{v},\alpha} \leq \delta$ (in fact, we have $\sum_\alpha \lambda_{\mathbf{u},\alpha} \leq \delta$). Therefore, we can now compute the edit distance as follows:

$$\mathsf{ED}(\mathsf{Share}_{\mathbf{u}}, \mathsf{Share}_{\mathbf{v}}) = \frac{1}{2}\sum_\alpha \mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|] \leq \frac{N}{2} \cdot \sqrt{\frac{8\delta}{N}} = \sqrt{2\delta N}.$$

□

Combining this with the result from Lemma 3.5.2, we can now compute the expected edit distance when $\mathbf{u}$ and $\mathbf{v}$ follow a distribution instead of being fixed.

$$\begin{aligned}
\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}_{\mathbf{v}\ominus\mathbf{u}})\right] &\leq \sqrt{2N} \cdot \mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\sqrt{\mathsf{ED}(\mathbf{u},\mathbf{v})}\right] \\
&\leq \sqrt{2N}\left(\frac{BN}{2}\right)^{1/4} = (2)^{1/4}B^{1/4}(N)^{3/4}
\end{aligned}$$

where the second step is by the concave Jensen's inequality.

C.3.3 Bounding the final statistical distance

Before we bound the final statistical distance, we introduce a useful result from Boyle et al. [BGIK22].

Lemma C.3.3 ([BGIK22])**.** *Consider ℓ balls thrown into N bins (labeled using $[N]$ without loss of generality) independently and uniformly at random. Let $\mathcal{U}_\alpha$ denote the final distribution of the configuration after another ball is added into bin α. Then, for all bins α and α', $\mathsf{SD}(\mathcal{U}_\alpha, \mathcal{U}_{\alpha'}) \leq \sqrt{\frac{N}{\ell}}$.*

While the original result in [BGIK22] is stated in terms of removing a ball either from bin α or α', we note that our formulation is equivalent since the statistical distance does not change by adding the same balls (one each in the two bins) to both distributions.

A more general bound for adding δ balls can also easily be derived. Suppose that we use the notation $S_\ell(\delta)$ to denote the maximum statistical distance when δ balls are added after throwing ℓ balls independently and uniformly at random. In particular, for $\Upsilon = (\upsilon_1, \ldots, \upsilon_\delta) \in [N]^\delta$, let $\mathcal{U}_\Upsilon$ denote the distribution when after throwing ℓ balls, a ball is added to each bin υ_i; Then $S_\ell(\delta) = \max_{\Upsilon,\Upsilon'} \mathsf{SD}(\mathcal{U}_\Upsilon, \mathcal{U}_{\Upsilon'})$. By hopping one ball at a time, we can use Lemma C.3.3 to directly conclude that $S_\ell(\delta) \leq \delta \cdot \sqrt{N/\ell}$.

We are now ready to bound the final statistical distance. Applying Markov's inequality to the result from the previous section, we get:

$$\Pr_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\mathsf{ED}(\mathsf{Share}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}_{\mathbf{v}\ominus\mathbf{u}}) \geq \gamma \cdot B^{1/4+\tau} N^{1/8}\right] \leq \frac{\sqrt[4]{2} \cdot B^{1/4} N^{3/4}}{\gamma \cdot B^{1/4+\tau} N^{1/8}} \leq \frac{2 \cdot N^{5/8}}{\gamma \cdot B^{\tau}}.$$

Define this probability as ρ_τ. As the final step, we can now use Lemma C.3.1 and Lemma C.3.3 to

compute the final statistical distance. Using $\tau = 1/8$ and $\ell = \gamma^4 B$, we get:

$$\begin{aligned}
\mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}}, \mathsf{Share}^{\ell}_{\mathbf{v}}) &\leq \mathsf{SD}(\mathsf{Share}^{\ell}_{\mathbf{u}\ominus\mathbf{v}}, \mathsf{Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}}) \\
&\leq (1-\rho_\tau)\cdot S_\ell(B^{1/4+\tau}N^{1/8}) + \rho_\tau \cdot S_\ell(B) \\
&\leq 1 \cdot \gamma B^{1/4+\tau}N^{1/8}\cdot\sqrt{\frac{N}{\ell}} + \frac{2\cdot N^{5/8}}{\gamma B^{\tau}}\cdot 1 \\
&= \frac{N^{5/8}}{\gamma B^{1/4-\tau}} + \frac{2\cdot N^{5/8}}{\gamma B^{\tau}} \leq \frac{3\cdot N^{5/8}}{\gamma B^{1/8}}.
\end{aligned}$$

Casting this back to our PIR context, since we have $N = |\mathcal{Q}_{\mathsf{IPIR}}|$ bins and $B = kC$ balls, and $\ell = B$, the final statistical distance is bounded by $\frac{3|\mathcal{Q}_{\mathsf{IPIR}}|^{5/8}}{(kC)^{1/8}}$. Recall that in Section 3.5.4, we can make $|\mathcal{Q}_{\mathsf{IPIR}}| = \Theta(n)$, where n is the database size. Therefore, for all $\epsilon = \epsilon(n) \geq 0$, there exists a constant d such that given $C \geq \frac{1}{\epsilon^8}\cdot\frac{dn^5}{k}$ honest clients queries, the statistical distance is bounded by ϵ.

Notice that there is an interesting trade-off between the number of clients required and the random noise used per client. By having each client provide γ times more noise queries, the statistical distance is reduced by a factor of $\sqrt[4]{\gamma}$, which in turn reduces the number of clients required by a factor of $(\sqrt[4]{\gamma})^8 = \gamma^2$.

C.3.4 Cost analysis

As in Theorem 3.5.1, k, Q, A are all functions of n. Below we write e.g., $k(n)$ as k for simplicity. To analyze the cost, we need to first analyze the size of x' in Construction 3.5.1. Let σ be the size of x'.

- When the OPIR servers have different Answer algorithms, the IPIR database x' has $k\cdot Q$ entries, each of A bits. Here $\sigma = kQ$.
- When the OPIR servers have the same Answer algorithm, the size x' is simply Q, each entry of x' is of A bits. Here $\sigma = Q$.

Per-query communication. To issue a query to the original n-bit database, the client sends $3k$ messages in total ($2k$ messages for shares of OPIR sub-queries and k dummies). Each message is an IPIR sub-query, therefore the query size for ShPIR is $O(k\cdot\log\sigma)$, and the answer size is $O(kA\sigma^{1/2})$.

The communication cost is dominated by the answer size, hence $O(kA\sigma^{1/2})$. When $\sigma = kQ$, the communication is $O(k^{3/2} \cdot A \cdot Q^{1/2})$; when $\sigma = Q$, the communication is $O(k \cdot A \cdot Q^{1/2})$.

Per-query computation. Assuming preprocessing, the server computation is the number of bits it reads, which is simply the answer size. So the computation is the same as above.

Server storage. To preprocess a size-σ database in the two-server additive PIR protocol (Figure 3.2.1), the server chooses the parameter m' for IPIR and a constant c such that $m' = c \cdot \log \sigma$. So the sub-query space of IPIR, namely $\mathcal{Q}_{\mathsf{IPIR}}$, has size $2^{m'} = \sigma^c$ (and consequently the number of entries in the lookup table). We can in fact use a more fine-grained choice of m', so that the size of $\mathcal{Q}_{\mathsf{IPIR}}$ is $\widetilde{O}(\sigma)$; we provide details in Appendix C.5.

Each entry in the lookup table is an answer polynomial with the number of bits $A\sigma^{1/2}$. Putting these together, the server storage, including the preprocessing bits, is $\widetilde{O}(A \cdot \sigma^{3/2})$. If $\sigma = kQ$, then the storage is $\widetilde{O}(A \cdot k^{3/2} \cdot Q^{3/2})$; if $\sigma = Q$, then the storage is $\widetilde{O}(A \cdot Q^{3/2})$.

C.4 Complete security proof for s-CNF-ShPIR (Theorem 3.5.3)

We now provide details for analyzing the construction where CNF-sharing is used for IPIR instead of 2-additive sharing. The basic structure of the proof is quite similar; notice that among the three major proof steps for Theorem 3.5.1, only the second part needs to be changed to reflect the CNF-sharing. This essentially requires analysis on how the balls in a configuration $\mathbf{u}$ get split into new balls corresponding to the CNF shares.

Definition C.4.1 (Cyclic rotations)**.** For a vector $\alpha = (\alpha_1, \ldots, \alpha_s)$, define its γ-cyclic rotation $(0 \le \gamma < s)$ as the vector $\alpha^{(\gamma)} = (\alpha_{\gamma+1}, \ldots, \alpha_s, \alpha_1, \ldots, \alpha_\gamma)$ where α_0 is defined to be α_s.

CNF-sharing details. Consider a (δ, N)-valid configuration $\mathbf{u}$ where the bins are labeled using elements in $\mathbb{G}$. For a given ball b, the s-CNF sharing procedure is as follows: First b is randomly split into s additive shares $\beta = (\beta_1, \ldots, \beta_s)$; i.e., $\beta_1, \ldots, \beta_{s-1}$ are first independently and uniformly sampled from $\mathbb{G}$, and then β_s is set to $b - \sum_{i=0}^{s-1} \beta_i$. Now, the s-CNF shares are defined to the cyclic rotations of β where the last element is dropped. In particular, the CNF-shares of β are $\alpha^{(0)}, \ldots, \alpha^{(s-1)}$ where $\alpha^{(i)} = (\beta_{i+1}, \ldots, \beta_s, \beta_1, \ldots, \beta_{i-1})$ and β_0 is defined to β_s.

C.4.1 Balls-and-bins analysis for CNF-shares

Notice that CNF-share is a vector in $\mathbb{G}^{s-1}$, and consequently, there are N^{s-1} bins within which the ball corresponding to each CNF-share can lie. Our goal now, very abstractly, is to understand the conditions under which one (or more) of the s balls corresponding to the CNF-shares resultant from splitting a ball b in $\mathbf{u}$ fall into a particular bin $\alpha \in \mathbb{G}^{s-1}$. This involves taking into account the symmetries of the CNF-shares towards which, we introduce some useful definitions.

Definition C.4.2 (Cyclic symmetries)**.** For a vector $\alpha = (\alpha_1, \ldots, \alpha_s)$, define the number of cyclic symmetries of α, denoted by $\mathsf{SymCyc}(\alpha)$, as the number of cyclic rotations $\alpha^{(\gamma)}$ where $(0 \leq \gamma < s)$ that are equal to α. Further, define the number of distinct cyclic rotations, denoted by $\mathsf{DistCyc}(\alpha)$, as the cardinality of the set $\{\alpha^{(\gamma)} \mid 0 \leq \gamma < s\}$.

Lemma C.4.3. *For any $\alpha = (\alpha_1, \ldots, \alpha_s)$, it holds that $\mathsf{SymCyc}(\alpha) \cdot \mathsf{DistCyc}(\alpha) = s$.*

Proof. The proof is quite straightforward using a group theoretic formulation. Notice that the group of cyclic rotations of is isomorphic to the group $\mathbb{Z}_s$ under addition modulo s; intuitively $\gamma \in \mathbb{Z}_s$ will correspond to a γ-cyclic rotation. Let c be the smallest positive integer such that $\alpha^{(c \bmod s)} = \alpha$. Then for all $\alpha^{(\gamma)} = \alpha$, notice that $\gamma \in \langle c \rangle$ (the subgroup of $\mathbb{Z}_s$ generated by c) which is therefore of size exactly $\mathsf{SymCyc}(\alpha)$. Further, the number of cosets of $\langle c \rangle$ in $\mathbb{Z}_s$ is exactly the number of distinct cyclic rotations $\mathsf{DistCyc}(\alpha)$. Therefore, by Lagrange's theorem, we directly have $\mathsf{SymCyc}(\alpha) \cdot \mathsf{DistCyc}(\alpha) = s$. □

Now, coming back to the CNF-sharing problem at hand, consider a (δ, N)-valid configuration $\mathbf{u}$. Define $s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}}$ to be the distribution of the balls-and-bins configuration when each ball in $\mathbf{u}$ is split into s-CNF shares. Note that we only need to consider (δ, N)-configurations since the proof of Lemma C.3.1 also directly works for $s\text{-}\mathsf{CNF\text{-}Share}$. We now show the following lemma.

Lemma C.4.4. *Consider a (δ, N)-valid configurations $\mathbf{u}$ and $\mathbf{v}$. Then,*

$$\mathsf{ED}(s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}}, s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{v}}) \leq sN^{(s-1)/2}\sqrt{\delta}.$$

Proof. Analyzing s-$\mathsf{CNF\text{-}Share}_{\mathbf{u}}$ essentially boils down to two parts:

1. What is the probability that that a ball b will lead to a CNF-share α (or equivalently, a ball in bin α within s-$\mathsf{CNF\text{-}Share}_{\mathbf{u}}$) (notice that the random variable will be $\mathsf{Bernoulli}$ and so we only need to find the probability).

2. Due to the symmetries of CNF-sharing, when one CNF-share is α, how many more shares will also be exactly α? In other words, if a ball lands in bin α, does this force any other balls to also land in α? (for instance, in the 2-additive sharing, when we had $b = 2\alpha$ for a ball b and bin α, if one additive share was α, then the other share would also be α).

Let $\mathbb{U}_\alpha$ represent the random variable for the number of balls in bin α for the distribution s-$\mathsf{CNF\text{-}Share}_{\mathbf{u}}$. As in the proof for Theorem 3.5.4, we wish to find $\mathbb{E}[\mathbb{U}_\alpha]$ and $\mathrm{Var}[\mathbb{U}_\alpha]$ and use them to bound the edit distance between s-$\mathsf{CNF\text{-}Share}_{\mathbf{u}}$ and s-$\mathsf{CNF\text{-}Share}_{\mathbf{v}}$ for any two $\mathbf{u}$ and $\mathbf{v}$.

For $1 \leq \tau \leq s$ and $\alpha \in \mathbb{G}^{s-1}$, let $\lambda_{\mathbf{u},\tau,\alpha}$ denote the number of balls b_i in $\mathbf{u}$ such that for the vector $\alpha^* = (\alpha_1, \ldots, \alpha_{s-1}, b_i - \sum_i \alpha_i)$, it holds that $\mathsf{SymCyc}(\alpha^*) = \tau$, and consequently $\mathsf{DistCyc}(\alpha^*) = s/\tau$ (from Lemma C.4.3). First notice that $\sum_\tau \lambda_{\mathbf{u},\tau,\alpha} = \delta$ since each ball will in some $\mathsf{SymCyc}(\alpha)$ value from 0 to s.

Now, $\mathsf{SymCyc}(\alpha^*)$ exactly corresponds to the number of CNF-shares that will fall into bin α if one CNF-share for the ball b_i is α. In addition, the probability that a CNF-share for b_i is α is exactly the number of distinct cyclic rotations divided by the number of bins, i.e., $\frac{\mathsf{DistCyc}(\alpha^*)}{N^{s-1}}$.

The number of balls added to bin α by each CNF-share of a ball in $\mathbf{u}$ is a Bernoulli random variable with probability $\frac{\mathsf{DistCyc}(\alpha^*)}{N^{s-1}}$; $\mathbb{U}_\alpha$ is just the sum of all these Bernoulli random variables. However, the symmetries of the CNF-sharing will create dependence between these random variables; this happens exactly for $\mathsf{SymCyc}(\alpha^*)$ number of variables, leading to their sum being distributed as $\mathsf{SymCyc}(\alpha^*)\mathsf{Ber} \cdot \left(\frac{\mathsf{DistCyc}(\alpha^*)}{N^{s-1}}\right)$. After accounting for these symmetries, the rest of the random variables are all independent.

We can now add all the Bernoulli random variables corresponding to all the balls b_i that result in

the same $\mathsf{DistCyc}(\alpha^*)$ (which from Lemma C.4.3 also means the same $\mathsf{SymCyc}(\alpha^*)$) to get a binomial distribution with the same probability. Consequently, the distribution of $\mathbb{U}_\alpha$ can be given by:

$$\mathbb{U}_\alpha \sim \sum_\tau \tau \cdot \mathsf{Binom}(\lambda_{\mathbf{u},\tau,\alpha}, \frac{s/\tau}{N^{s-1}}).$$

Notice that this also cleanly captures the distribution resultant from the two-additive sharing. Now, we can compute the expectation and variance as:

$$\mathbb{E}[\mathbb{U}_\alpha] = \sum_\tau \frac{s\lambda_{\mathbf{u},\tau,\alpha}}{N^{s-1}} = \frac{s\delta}{N^{s-1}}$$

$$\begin{aligned}
\mathrm{Var}[\mathbb{U}_\alpha] &= \sum_\tau \tau^2 \cdot \lambda_{\mathbf{u},\tau,\alpha} \cdot \frac{s}{\tau N^{s-1}} \cdot \left(1 - \frac{s}{\tau N^{s-1}}\right) \\
&\leq \frac{s}{N^{s-1}} \sum_\tau \tau \lambda_{\mathbf{u},\tau,\alpha} \cdot 1 \\
&\leq \frac{s}{N^{s-1}} \cdot s\delta = \frac{s^2\delta}{N^{s-1}}
\end{aligned}$$

since $\sum_\tau \tau\lambda_{\mathbf{u},\tau,\alpha}$ is maximized when $\lambda_{\mathbf{u},s,\alpha} = \delta$ and the other $\lambda_{\mathbf{u},\tau\neq s,\alpha} = 0$.

Similarly, for any other another (δ, N)-valid $\mathbf{v}$, we can compute:

$$\mathbb{E}[\mathbb{V}_\alpha] = \frac{s\delta}{N^{s-1}} \qquad \text{and} \qquad \mathrm{Var}[\mathbb{V}_\alpha] \leq \frac{s^2\delta}{N^{s-1}}$$

Now applying the Jensen's inequality $\mathbb{E}[Z] \leq \sqrt{\mathbb{E}[Z^2]}$ to the random variable $|\mathbb{U}_\alpha - \mathbb{V}_\alpha|$, we get:

$$\begin{aligned}
\mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|] &\leq \sqrt{\mathbb{E}[(\mathbb{U}_\alpha - \mathbb{V}_\alpha)^2]} = \sqrt{\mathbb{E}[(\mathbb{U}_\alpha)^2] + \mathbb{E}[(\mathbb{V}_\alpha)^2] - 2 \cdot \mathbb{E}[\mathbb{U}_\alpha] \cdot \mathbb{E}[\mathbb{U}_\alpha]} \\
&\leq \sqrt{\left(\frac{s\delta}{N^{s-1}}\right)^2 + \frac{s^2\delta}{N^{s-1}} + \left(\frac{s\delta}{N^{s-1}}\right)^2 + \frac{s^2\delta}{N^{s-1}} - \frac{2s^2\delta^2}{N^{2s-2}}} \\
&= \sqrt{\frac{2s^2\delta}{N^{s-1}}} = \frac{\sqrt{2}s}{N^{(s-1)/2}} \cdot \sqrt{\delta}.
\end{aligned}$$

Therefore, we can now compute the edit distance as follows:

$$\begin{aligned}\mathsf{ED}(s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}}, s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{v}}) &= \frac{1}{2}\sum_{\alpha}\mathbb{E}[|\mathbb{U}_\alpha - \mathbb{V}_\alpha|] \\ &\leq \frac{N^{s-1}}{2}\cdot\frac{\sqrt{2}s}{N^{(s-1)/2}}\cdot\sqrt{\delta} \\ &\leq sN^{(s-1)/2}\sqrt{\delta}.\end{aligned}$$

□

Combining this with the result from Lemma 3.5.2, we can now compute the expected edit distance when $\mathbf{u}$ and $\mathbf{v}$ follow a distribution instead of being fixed.

$$\begin{aligned}\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\mathsf{ED}(s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}\ominus\mathbf{v}}, s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{v}\ominus\mathbf{u}})\right] &\leq \sqrt{2N}\cdot\mathbb{E}_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}\left[\sqrt{\mathsf{ED}(\mathbf{u},\mathbf{v})}\right] \\ &\leq sN^{(s-1)/2}\left(\frac{BN}{2}\right)^{1/4} \\ &\leq sN^{(2s-1)/4}B^{1/4}.\end{aligned}$$

where the second step is by the concave Jensen's inequality. Now, using Markov's inequality, we get

$$\Pr_{\mathbf{u}\sim\mathcal{B},\mathbf{v}\sim\mathcal{B}'}[\mathsf{ED}(s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{u}\ominus\mathbf{v}}, s\text{-}\mathsf{CNF\text{-}Share}_{\mathbf{v}\ominus\mathbf{u}}) \geq \sqrt{s}N^{(2s-3)/8}B^{1/4+\tau}] \leq \frac{\sqrt{s}\cdot N^{(2s+1)/8}}{B^{1/4+\tau}}.$$

Define this probability as ρ'_τ. Taking the total number of extra balls $\ell = B$, and $\tau = 1/8$,

$$\begin{aligned}\mathsf{SD}(s\text{-}\mathsf{CNF\text{-}Share}^{\ell}_{\mathbf{u}}, s\text{-}\mathsf{CNF\text{-}Share}^{\ell}_{\mathbf{v}}) &\leq \mathsf{SD}(s\text{-}\mathsf{CNF\text{-}Share}^{\ell}_{\mathbf{u}\ominus\mathbf{v}}, s\text{-}\mathsf{CNF\text{-}Share}^{\ell}_{\mathbf{v}\ominus\mathbf{u}}) \\ &\leq (1-\rho'_\tau)\cdot S_\ell(\sqrt{s}N^{(2s-3)/8}B^{1/4+\tau}) + \rho'_\tau\cdot S_\ell(sB) \\ &\leq 1\cdot\sqrt{s}N^{(2s-3)/8}B^{1/4+\tau}\cdot\sqrt{\frac{N}{\ell}} + \frac{\sqrt{s}\cdot N^{(2s+1)/8}}{B^{1/4+\tau}}\cdot 1 \\ &= \frac{\sqrt{s}\cdot N^{(2s+1)/8}}{B^{1/4-\tau}} + \frac{\sqrt{s}\cdot N^{(2s+1)/8}}{B^{\tau}} \leq \frac{2\sqrt{s}\cdot N^{(2s+1)/8}}{B^{1/8}}.\end{aligned}$$

C.5 Proof for the concrete construction (Theorem 3.5.4)

Before we give the full proof, we first prove a small lemma below, which provides a way to bound the size of sub-query space in 2-additive PIR within polylogarithmic overhead of the database size.

Lemma C.5.1. *For any $n \in \mathbb{N}$ and $n \geq 4$, there always exists a constant c^* such that*

$$\binom{\log n + c^* \log\log n + 1}{(\log n + c^* \log\log n + 1)/2} \geq n.$$

Proof. By Stirling formula, we have

$$\binom{\log n + c^* \log\log n + 1}{(\log n + c^* \log\log n + 1)/2} \geq \frac{2\sqrt{2\pi}}{e^2} \cdot \frac{2^{\log n + c^* \log\log n + 1}}{\sqrt{\log n + c^* \log\log n + 1}}.$$

To ensure the equation in the lemma holds, it is sufficient to ensure

$$\frac{n \cdot (\log n)^c}{\sqrt{\log n + c^* \log\log n + 1}} \geq n.$$

Following above, it is sufficient to ensure

$$(\log n)^{c^*} > 3 \log n,$$

and we know this is equivalent to $c^* > \frac{1}{2} \cdot (1 + \frac{\log 3}{\log\log n})$. Assume $n \geq 4$, then such constant c^* exists. □

Cost analysis. Following the parameter choice specified in Section 3.5.4, we already have the set of parameters for OPIR and IPIR that compile. The only thing left is to choose $k, m, d, t, |\mathbb{F}|$ for OPIR (Reed-Muller PIR) and s, m', d' for IPIR (CNF PIR) based on a given constant γ.

IPIR *database size.* First, let $m = 2/\gamma$, then k and $|\mathbb{F}|$ are both $O(n^{\gamma/2})$. According to parameter choice specified in Section 3.5.4, the IPIR database x' (before preprocessing) has the number of entries $\Theta(n)$.

IPIR *preprocessing.* Choose $s = 2/\gamma$. Each answer in IPIR consists of $O(n^{1/s})$ monomials with coefficients represented by $\log |\mathbb{F}|$ bits, so it has the number of bits $O(n^{\gamma/2} \log n)$.

Now we want to bound the number of entries in the lookup table in the IPIR preprocessing. Let c^* be a constant; we choose $m' = \log n' + c^* \log\log n' + 1$ and $d' = m'/2 = (\log n' + c^* \log\log n' + 1)/2$. By Lemma C.5.1, the choice of m' results in $|\mathcal{Q}_{\mathsf{IPIR}}| = 2^{m'} = \widetilde{O}(n)$; this is also the entries in the lookup table. Plug in the answer size of IPIR above, the total number of preprocessing bits (i.e., the server storage) is $\widetilde{O}(n^{1+\gamma/2})$.

Communication and computation. For each query, the client sends $k \cdot (s+1)$ messages, each message of $\Theta(\log n)$ bits. Therefore the query size of ShPIR is $O(n^{\gamma/2} \log n)$. The answer to a query consists of $k \cdot (s+1) \cdot s$ messages, each message of $O(n^{\gamma/2} \log n)$ bits; so the answer size of ShPIR is $O(n^{\gamma} \log n)$. Since answering each query is just a table lookup, the number of bits that the server needs to read (computation cost) is exactly the same as the answer size.

A final complication is that we can get rid of the $\log n$ term by choosing a constant $\frac{2}{2/\gamma - \gamma/2 + 1} < \gamma' < \gamma$, and then set all parameters as above using γ' instead of γ. This results in per-query communication and computation both $O(n^{\gamma})$, and the server storage is $O(n^{\gamma'/2 + 2/\gamma' - 1})$, which is bounded by $O(n^{2/\gamma})$ since $\gamma'/2 + 2/\gamma' - 1 < \gamma/2 + 2/\gamma' - 1 < 2/\gamma$ given the restrictions on γ' as above.

Total number of queries for security. We use Theorem 3.5.3 and plug in parameters for OPIR. Select parameters as described in Section 3.5.4, and let $m = \gamma/2$ and $s = 2/\gamma$ as above, then the $Q = |\mathcal{Q}_{\mathsf{OPIR}}| = c_1 \cdot n$, and $k = c_2 \cdot n^{\gamma/2}$ for where c_1, c_2 are constant. Using Theorem 3.5.3, we have $Q^{2s+1}/k\epsilon^8 = (c_1^{2s+1}/c_2^{\gamma/2}) \cdot n^{4/\gamma - \gamma/2 + 1}/\epsilon^8$. Let $c_0 = (c_1^{4/\gamma+1}/c_2^{\gamma/2})$, we have proved that for all $C \geq c_0 n^{4/\gamma+1}/\epsilon^8$, the composed construction has security ϵ.

The final complication is that we need to choose $\gamma' < \gamma$ in terms of efficiency (as we did for the communication cost above), therefore the resulting term is $c \cdot n^{4/\gamma' - \gamma'/2 + 1}$. This is asymptotically smaller than $n^{4/\gamma}$ if we choose $\gamma' < \gamma$ such that $4/\gamma' - \gamma'/2 < 4/\gamma$. Note that the restriction on γ' is equivalent to $g(\gamma') = \gamma \cdot \gamma'^2 + 8\gamma' - 8\gamma > 0$, and such γ' exists because $g(\gamma) = \gamma^3 > 0$ and g is

continuous.

C.6 Deferred Material on PIR with Variable-Sized Records (Section 3.6)

Trade-off between security and efficiency. If c is 0, then in the construction in Figure 3.6.1, one can easily tell between the two input configurations, say (32, 2) and (16, 18). This is because with probability $1/2$, there is one ball at level 5 for the former case, while in the latter case there is always no ball at level 5. In fact, a more generic example is $(S-C+1, 1, 1, \ldots, 1)$ and $(S/C, S/C, \ldots, S/C)$ (both cases sums up to S), and the latter always has no ball at the level higher than $\log L - \log C$.

Therefore, we fully split the balls at the first few levels; and then do the probabilistic splitting. We call the highest level that has probabilistic splitting as the *full-split level*; the number of levels where balls are fully split is *depth* of the full-split level. We next show that, if there are many balls at the full-split level, then after the probabilistic splitting, those balls can "smooth out" the different configurations at the lower levels.

In our construction (Figure 3.6.1), we need $\log C < \rho < c \log L$ for constant $0 < c < 1$. Let $d = \rho / \log C$; then from above C^d should be much smaller than L. Therefore, we restrict C to be $O(\mathsf{polylog} L)$ for the of analysis of our construction.

Proof of Theorem 3.6.2

Same as our proof of Theorem 3.5.1 (Section 3.5.1), the shuffler completely eliminates the order of sub-lengths. Therefore, the security analysis reduces to understanding the distribution of balls over the $\log L$ levels; again we view this as a balls-to-bins problem.

Our proof consists of two steps. First, we show that if there are sufficiently many balls at the highest level, these balls can "smooth out" the differences in the lower levels after the recursive splitting, even if the configurations of the lower levels are different. More formally, given two different configurations at level lower than j, place k balls at level j, then after the splitting, the resulting distributions are statistically close if the the edit distance between the two configurations is much smaller than k.

Next we show that, for any input configurations, at the full-split level, there are sufficiently many balls there. In particular, we prove that the minimum number of balls one can have is not much smaller than the maximum possible number of balls, conditioning on the sum of all the configurations being equal. And combined with a special configuration where there are 2^ρ balls at the full-split level, this completes our proof.

Lemma C.6.1 (Smoothing Lemma)**.** *Given k, r such that $0 < r < \sqrt{k}$, consider placing k balls at level j v.s. placing $k - r$ balls at level j and place $2r$ balls at level $j - 1$. Let $\mathcal{D}_0$ be the distribution of balls after randomized splitting starting with the first configuration; similarly $\mathcal{D}_r$ for the second configuration. Then there exists a constant $c > 0$ such that*

$$\mathsf{SD}(\mathcal{D}_0, \mathcal{D}_r) \leq c \cdot r/\sqrt{k}.$$

Proof. We start with a simple case where $r = 1$.

$$\begin{aligned}
\mathsf{SD}(\mathcal{D}_0, \mathcal{D}_1) \leq & (\frac{1}{2})^k \\
& + \left| \binom{k}{1} (\frac{1}{2})^k - \binom{k-1}{0} (\frac{1}{2})^{k-1} \right| \\
& + \left| \binom{k}{2} (\frac{1}{2})^k - \binom{k-1}{1} (\frac{1}{2})^{k-1} \right| \\
& + \dots \\
& + \left| \binom{k}{k-1} (\frac{1}{2})^k - \binom{k-1}{k-2} (\frac{1}{2})^{k-1} \right| \\
& + \left| \binom{k}{k} (\frac{1}{2})^k - \binom{k-1}{k-1} (\frac{1}{2})^{k-1} \right| \\
\leq & \binom{k}{k/2} (\frac{1}{2})^k = \Theta(\frac{1}{\sqrt{k}}).
\end{aligned}$$

The same idea applies to any $r < \sqrt{k}$. We have

$$
\begin{aligned}
\mathsf{SD}(\mathcal{D}_0, \mathcal{D}_r) \leq & (\frac{1}{2})^k \cdot (\binom{k}{0} + \ldots + \binom{k}{r}) \\
& + \left| \binom{k}{r} (\frac{1}{2})^k - \binom{k-r}{0} (\frac{1}{2})^{k-r} \right| \\
& + \left| \binom{k}{r+1} (\frac{1}{2})^k - \binom{k-r}{1} (\frac{1}{2})^{k-r} \right| \\
& + \ldots \\
& + \left| \binom{k}{k} (\frac{1}{2})^k - \binom{k-r}{k-r} (\frac{1}{2})^{k-r} \right| \\
& \leq 2r \cdot \binom{k}{k/2} (\frac{1}{2})^k \leq c \cdot \frac{2r}{\sqrt{k}} \text{ for some constant } c > 0.
\end{aligned}
$$

□

Lemma C.6.2 (Bounding the differences at every level)**.** *Given any two configurations with equal sum, then at every level, the maximum number of balls and the minimum number of balls differ at most C.*

Proof. Let the sum be S and the number of clients be C. Now consider the balls-to-bins configuration of all the clients.

The maximum number of balls above the k-th level is $C(2^{k-1} + \ldots 2^0) = C(2^k - 1)$, as for each client, each level has at most one ball. So in this case, the number of balls at the k-th level is $\frac{S - C(2^k - 1)}{2^k} = \frac{S}{2^k} - C + \frac{C}{2^k}$.

In the other case, the minimum number of balls below the k-th level is 0; then in this case the k-th level has the number of balls $S/2^k$.

Therefore, the difference is $C - \frac{C}{2^k}$, which is smaller than C. □

Analysis of message complexity. Now we can analyze, for a given security ϵ, and C clients, what is trade-off ρ we need (this affects the message complexity). Note that here we analyze for a single client; this is different from the all-client case in the proof of Lemma C.6.2 above.

As the analysis in the beginning of Appendix C.6, let $d = \rho / \log C$. By union bound, we can derive the relation between a target security error ϵ and the depth of full-split level ρ. That is, $(\log L \cdot \log C)/\sqrt{C^d} \leq \epsilon$, where a sufficient equation is $C^{d-1} \geq 1/\epsilon^2$.

Now we want to compute the message complexity. Suppose the record is of length ℓ, if there is only one ball at level $\log \ell$, then in expectation there will be $\log \ell$ balls in the end. Also, each ball at a given level is independently split. So we only need to analyze how many balls fall in the full-split level.

- If $\log \ell < \log L - \rho$, namely the record length is small, then from above we immediately get the message complexity being $\log \ell$.

- If $\log \ell \geq \log L - \rho$, then all the balls between the $(\log L - \rho)$-th level and the $(\log \ell)$-th level have to be fully split. This means there are at most $2\ell C^d / L$ balls land in the full-split level. Therefore, in expectation there are $2\ell C^d \log \ell / L \leq 2C^d \log \ell$ balls at the end. Recall that to ensure security ϵ, we need $C^{d-1} \geq 1/\epsilon^2$. Combining the two inequalities, we have message complexity $h \geq (2C \cdot \log \ell)/\epsilon^2$. When $C = \mathsf{polylog} L$, then the overhead (message complexity) is just $\mathsf{polylog} L/\epsilon^2$; note that this is also $\mathsf{polylog} \ell/\epsilon^2$.

BIBLIOGRAPHY

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. Technical report, 2015. https://www.tensorflow.org/.

[ABIW22] Sebastian Angel, Andrew J. Blumberg, Eleftherios Ioannidis, and Jess Woods. Efficient representation of numerical optimization problems for SNARKs. In *Proceedings of the USENIX Security Symposium*, 2022.

[ACD+21] Erik Anderson, Melissa Chase, Wei Dai, F. Betul Durak, Kim Laine, Siddhart Sharma, and Chenkai Weng. Aggregate measurement via oblivious shuffling. Cryptology ePrint Archive, Paper 2021/1490, 2021. https://eprint.iacr.org/2021/1490.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 962–979, 2018.

[AGJ+22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In Clemente Galdi and Stanislaw Jarecki, editors, *Proceedings of the International Conference on Security and Cryptography for Networks(SCN)*, 2022.

[AIK07] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. In *Annual International Cryptology Conference*, pages 92–110. Springer, 2007.

[AIK+21] Shweta Agrawal, Yuval Ishai, Eyal Kushilevitz, Varun Narayanan, Manoj Prabhakaran, Vinod M. Prabhakaran, and Alon Rosen. Secure computation from one-way noisy communication, or: Anti-correlation via anti-concentration. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 124–154, 2021.

[AIVG22] Kinan Dak Albab, Rawane Issa, Mayank Varia, and Kalman Graffi. Batched Differentially Private Information Retrieval. In *Proceedings of the USENIX Security Symposium*, pages 3327–3344, 2022.

[AJM+23] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern.

Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2023.

[ALP+21] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In *Proceedings of the USENIX Security Symposium*, pages 1811–1828, 2021.

[AMBFK16] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, July 2016.

[ANOS24] Bar Alon, Moni Naor, Eran Omri, and Uri Stemmer. Mpc for tech giants (gmpc): enabling gulliver and the lilliputians to cooperate amicably. In *Annual International Cryptology Conference*, pages 74–108. Springer, 2024.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3), October 2015.

[APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1233–1252, 2020.

[AS16] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, 2016.

[ASA+21] Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Coeus: A system for oblivious document ranking and retrieval. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[AYA+21] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

[BBCG+21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[BBDBM18] Benedikt Bunz, Jonathan Bootle, Pieter Wuille Dan Boneh, Andrew Poelstra, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[BBG+20] James Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana

Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[BBG23] Borja Balle, James Bell, and Adrià Gascón. Amplification by shuffling without shuffling, 2023. https://arxiv.org/pdf/2305.10867.pdf.

[BBGN20] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. Private summation in the multi-message shuffle model. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[BCGL+24] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Phillipp Schoppmann. Willow: Secure aggregation with one-shot clients. *Cryptology ePrint Archive*, 2024.

[BCI+10] Eric Brier, Jean-Sebastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2010.

[BEM+17] Andrea Bittau, Ulfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. PROCHLO: Strong Privacy for Analytics in the Crowd. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[BFKL93] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J Lipton. Cryptographic primitives based on hard learning problems. In *Annual international cryptology conference*, pages 278–291. Springer, 1993.

[BGI+14] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2014.

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable Distributed Point Functions. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2022.

[BGL+22] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. Acorn: Input validation for secure aggregation. Cryptology ePrint Archive, Paper 2022/1461, 2022. https://eprint.iacr.org/2022/1461.

[BGR98] Mihir Bellare, Juan A Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology—EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31–June 4, 1998 Proceedings 17*, pages 236–250. Springer, 1998.

[BHB20a] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. ABIDES: Agent-based interactive discrete event simulation environment. https://github.com/abides-sim/abides, 2020.

[BHB20b] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. ABIDES: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2020.

[BHK+24] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. Sprint: High-throughput robust distributed schnorr signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 62–91. Springer, 2024.

[BIK05] Amos Beimel, Yuval Ishai, and Eyal Kushilevitz. General constructions for information-theoretic private information retrieval. In *Journal of Computer and System Sciences*, 2005.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H.Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the Servers' Computation in Private Information Retrieval: PIR with Preprocessing. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2000.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2017.

[Boo17] Jonathan Bootle. Efficient multi-exponentiation, 2017. https://jbootle.github.io/Misc/pippenger.pdf.

[BVH+20] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov. How to backdoor federated learning. In *Proceedings of the Artificial Intelligence and Statistics Conference (AISTATS)*, 2020.

[Cao24] Dennis Cao. Ai helped the u.s. government recover $1 billion in fraud in 2024—and it's just getting started. https://cdotimes.com/2024/10/17/ai-helped-the-u-s-government-recover-1-billion-in-fraud-in-2024-and-its-just-getting-started/, 2024. Accessed: 2025-05-03.

[CCSL24] Lynn Chua, Hao Chen, Yongsoo Song, and Kristin Lauter. On the concrete security of lwe with small secret. *La Matematica*, 3(3):1032–1068, 2024.

[CD17] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[CDGM19] M. Chase, A. Deshpande, E. Ghosh, and H. Malvai. Seemless: Secure end-to-end encrypted messaging with less trust. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[CDW+] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. https://github.com/TalwalkarLab/leaf.

[CDW+18] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018. https://arxiv.org/abs/1812.01097.

[CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[CGHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2022.

[CGJvdM22] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1995.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. In *Communications of the ACM (CACM)*, 1981.

[Cha88] David L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1), 1988.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, October 2018.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient pri-

vate information retrieval. In *Proceedings of the Theory of Cryptography Conference (TCC)*, November 2017.

[CKK+21] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Joohee Lee, Junbum Shin, and Yongsoo Song. Lattice-based secure biometric authentication for hamming distance. In *Information Security and Privacy: 26th Australasian Conference, ACISP 2021, Virtual Event, December 1–3, 2021, Proceedings 26*, pages 653–672. Springer, 2021.

[CL15] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from ddh. Cryptology ePrint Archive, Paper 2015/047, 2015. https://eprint.iacr.org/2015/047.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1986.

[clo] Cloudflare randomness beacon. https://developers.cloudflare.com/randomness-beacon/.

[CS03] Don Coppersmith and Madhu Sudan. Reconstructing Curves in Three (and Higher) Dimensional Space from Noisy Data. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2003.

[CS04] John Canny and Stephen Sorkin. Practical large-scale distributed key generation. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004.

[CSS11] T-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-preserving stream aggregation with fault tolerance. In *Proceedings of the International Financial Cryptography Conference*, 2011.

[CSU+19] Albert Cheu, Adam D. Smith, Jonathan R. Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 375–403, 2019.

[CU21] Albert Cheu and Jonathan R. Ullman. The limits of pan privacy and shuffle privacy for learning and estimation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 1081–1094, 2021.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 1989.

[DFL+20] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. {DORY}: An encrypted search system with distributed trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119, 2020.

[DG15] Zeev Dvir and Sivakanth Gopi. 2-server pir with sub-polynomial communication. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2015.

[DKIR21] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2006.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the USENIX Security Symposium*, August 2004.

[DN03] Irit Dinur and Kobbi Nissim. Revealing Information while Preserving Privacy. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, 2003.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[DSDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. Lwe with side information: Attacks and concrete security estimation. In *Annual international cryptology conference*, pages 329–358. Springer, 2020.

[DSM22] Vishnu Asutosh Dasu, Sumanta Sarkar, and Kalikinkar Mandal. PROV-FL: Privacy-preserving round optimal verifiable federated learning. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, 2022.

[dVYA18] Henry de Valence, Cathie Yun, and Oleg Andreev. Bulletproof implementation based on delak-cryptography, 2018. https://github.com/dalek-cryptography/bulletproofs.

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[EDG14] Tariq Elahi, George Danezis, and Ian Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[EFM+20] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Shuang Song, Kunal Talwar, and Abhradeep Thakurta. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *CoRR*, abs/2001.03618, 2020.

[EZE+23] Ahmed Roushdy Elkordy, Jiang Zhang, Yahya H. Ezzeldin, Konstantinos Psounis, and

Salman Avestimehr. How Much Privacy Does Federated Learning with Secure Aggregation Guarantee? In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[Fed24] Federal Trade Commission. A look behind the scenes: Examining the data practices of social media and video streaming services. https://www.ftc.gov/system/files/ftc_gov/pdf/Social-Media-6b-Report-9-11-2024.pdf, 2024. Accessed: 2025-05-03.

[Fei99] Uriel Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 142–152. IEEE, 1999.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1987.

[Fen23] Boyuan Feng. Multi-scalar multiplication (MSM), 2023. https://hackmd.io/@tazAymRSQCGXTUKkbh1BAg/Sk27liTW9.

[FIC24] FICO. Td wins 2024 fico decisions award for its achievements in fraud management. https://www.fico.com/en/newsroom/td-wins-2024-fico-decisions-award-its-achievements-fraud-management, 2024. Accessed: 2025-05-03.

[FY92] Matthew Franklin and Moti Yung. Communication complexity of secure computation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, 1992.

[GCM+16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and Private Media Consumption with Popcorn. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once / secure MPC with stateless ephemeral roles. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2021.

[GHKR08] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold rsa for dynamic and adhoc groups. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2008.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 458–487. Springer, 2022.

[GHPS22] Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *Proceedings of the USENIX Security Symposium*, 2022.

[GIK+24] Adrià Gascón, Yuval Ishai, Mahimna Kelkar, Baiyu Li, Yiping Ma, and Mariana Raykova. Computationally secure aggregation and private information retrieval in the shuffle model. 2024.

[Gil59] E. N. Gilbert. Random graphs. In *The Annals of Mathematical Statistics*, 1959.

[GJKR06] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Journal of Cryptology*, 2006.

[GK10] Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2010.

[GKL+20] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. Pancake: Frequency smoothing for encrypted data stores. In *Proceedings of the USENIX Security Symposium*, 2020.

[GMPV20] Badih Ghazi, Pasin Manurangsi, Rasmus Pagh, and Ameya Velingker. Private Aggregation from Fewer Anonymous Messages. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[GPS+22] Yue Guo, Antigoni Polychroniadou, Elaine Shi, David Byrd, and Tucker Balch. MicroFedML: Privacy preserving federated learning for small weights. Cryptology ePrint Archive, Paper 2022/714, 2022. https://eprint.iacr.org/2022/714.

[HDCGZ23] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.

[Hen16] Ryan Henry. Polynomial batch codes for efficient IT-PIR. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, July 2016.

[HHCG+23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *Proceedings of the USENIX Security Symposium*, 2023.

[HHK+21] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. Merkle^2: A low-latency transparency log system. In *In Proceedings of the IEEE*

Symposium on Security and Privacy (S&P), 2021.

[HIKR23] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In *Annual International Cryptology Conference*, pages 203–235. Springer, 2023.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or how to cope with perpetual leakage. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 1995.

[HRX08] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 2008.

[HSS+21] Armando Faz Hernández, Samuel Scott, Nick Sullivan, Riad S. Wahby, and Christopher Wood. Hashing to elliptic curves. https://www.ietf.org/archive/id/draft-irtf-cfrg-hash-to-curve-10.html, 2021.

[HSSN+22] Kyle Hogan, Sacha Servan-Schreiber, Zachary Newman, Ben Weintraub, Cristina Nita-Rotaru, and Srinivas Devadas. Shortor: Improving tor network latency via multi-hop overlay routing. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[IKLM24] Yuval Ishai, Mahimna Kelkar, Daniel Lee, and Yiping Ma. Information-theoretic single-server PIR in the shuffle model. In *Information-Theoretic Cryptography (ITC) 2024*, 2024.

[IKOS04] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2004.

[IKOS06] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from Anonymity. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

[ISN87] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing schemes realizing general access structure. In *IEEE Global Telecommunication Conference*, 1987.

[JLS24] Aayush Jain, Huijia Lin, and Sagnik Saha. A systematic study of sparse lwe. In *Annual International Cryptology Conference*, pages 210–245. Springer, 2024.

[KEB98] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *Information Hiding*, 1998.

[KKMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and

threshold signatures. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2020.

[KLSS23] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-mlwe. In *Annual International Cryptology Conference*, pages 549–580. Springer, 2023.

[KMA+21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawit, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. In *Foundations and Trends in Machine Learning*, 2021.

[KMSW22] Ilan Komargodski, Shin'ichiro Matsuo, Elaine Shi, and Ke Wu. log*-round game-theoretically-fair leader election. In *Annual International Cryptology Conference*, pages 409–438. Springer, 2022.

[KMZ+19] Sai Krishna, Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: Dynamic-committee proactive secret sharing. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[KNH] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The CIFAR-100 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.

[KP24] Harish Karthikeyan and Antigoni Polychroniadou. Opa: one-shot private aggregation with single client interaction and its applications to federated learning. In *International Workshop on Federated Foundation Models in Conjunction with NeurIPS 2024*, 2024.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[KRKR20] Swanand Kadhe, Nived Rajaraman, O. Ozan Koyluoglu, and Kannan Ramchandran. FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning. In *ICML Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2020.

[LBV+23] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Robustness of secure federated learning. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[LGG+22] D. Leung, Y. Gilad, S. Gorbunov, L. Reyzin, and N. Zeldovich. Aardvark: An asynchronous authenticated dictionary with short proofs. 2022.

[Lip09] Helger Lipmaa. First CPIR protocol with data-dependent computation. In *ICISC*, 2009.

[LKK+18] Joohee Lee, Dongwoo Kim, Duhyeong Kim, Yongsoo Song, Junbum Shin, and Jung Hee Cheon. Instant privacy-preserving biometric authentication for hamming distance. *Cryptology ePrint Archive*, 2018.

[LLPT23] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. Lerna: secure single-server aggregation via key-homomorphic masking. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 302–334. Springer, 2023.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2023.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In *IACR International Conference on Public-Key Cryptography*, pages 215–241. Springer, 2021.

[LYK+19] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 887–903, 2019.

[LZMC21] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

[MBB+15] S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman. CONIKS: bringing key transparency to end users. 2015.

[MDC16] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics

with succinct sketches. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016.

[MMR+17] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Artificial Intelligence and Statistics Conference (AISTATS)*, 2017.

[MNS09] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2009.

[MOJC23] Mohamad Mansouri, Melek Onen, Wafa Ben Jaballah, and Mauro Conti. SoK: Secure aggregation based on cryptographic schemes for federated learning. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2023.

[MR23] Muhammad Haris Mughees and Ling Ren. Vectorized Batch Private Information Retrieval. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[MSCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[MW22] Samir Jordan Menon and David J. Wu. Spiral: Fast, High-Rate Single-Server PIR via FHE Composition. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2022.

[MWA+23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. Cryptology ePrint Archive, Paper 2023/486, 2023. https://eprint.iacr.org/2023/486.

[NLT24] Truong Son Nguyen, Tancrède Lepoint, and Ni Trieu. Mario: Multi-round multiple-aggregator secure aggregation with robustness against malicious actors. *Cryptology ePrint Archive*, 2024.

[PBB09] Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. VPriv: Protecting privacy in location-based vehicular services. 2009.

[PBBL11] Raluca Ada Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.

[Ped91] T. Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*

(EUROCRYPT), 1991.

[PFA22] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[Pip] Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptograpy. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, May 2005.

[RNFH19] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[RNM+21] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[RS86] Michael O Rabin and Jeffery O Shallit. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics*, 39(S1):S239–S256, 1986.

[RSWP23] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *In Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2023.

[RZHP20] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. Orchard: Differentially private analytics at scale. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[SAG+23] Jinhyun So, Ramy E Ali, Başak Güler, Jiantao Jiao, and A Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[SCR+11] Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2011.

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Journal of Cryptology*, 2002.

[SGA21] Jinhyun So, Basak Güler, and A. Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. In *Journal on Selected Areas in Information Theory*, 2021.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SHY+22] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E. Ali, Basak Guler, and Salman Avestimehr. LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning. In *Proceedings of Machine Learning and Systems*, 2022.

[SSV+22] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *Proceedings of the USENIX Security Symposium*, 2022.

[TBA+19] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the ACM workshop on artificial intelligence and security*, 2019.

[TBP+19] A. Tomescu, V. Bhupatiraju, D. Papadopoulos, C. Papamanthou, N. Triandopoulos, and S. Devadas. Transparency logs via append-only authenticated dictionaries. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2019.

[TDG16] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-Cost ϵ-Private Information Retrieval. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[TFZ+22] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. VeRSA: Verifiable registries with efficient client audits from RSA authenticated dictionaries. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[TKPS22] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2022.

[vdHLZZ15] Jelle van den Hoof, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2015.

[WB19] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems (CHES)*, 2019.

[WXWZ23] Lixu Wang, Shichao Xu, Xiao Wang, and Qi Zhu. Eavesdrop the composition proportion of training labels in federated learning. arXiv:1910/06044, 2023. https:

//arxiv.org/abs/1910.06044.

[WY05] David Woodruff and Sergey Yekhanin. A Geometric Approach to Information-Theoretic Private Information Retrieval. In *Computational Complexity Conference (CCC)*, 2005.

[YM20] Honglin Yuan and Tengyu Ma. Federated accelerated stochastic gradient descent. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[ZKP16] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *Proceedings of the USENIX Security Symposium*, 2016.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Neural Information Processing Systems (NeurIPS)*, 2019.

[ZMA22] Ke Zhong, Yiping Ma, and Sebastian Angel. Ibex: Privacy-preserving ad conversion tracking and bidding. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2022.

[ZMMA23] Ke Zhong, Yiping Ma, Yifeng Mao, and Sebastian Angel. Addax: A fast, private, and accountable ad exchange infrastructure. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation, 2023.

[ZSCM23] Mingxun Zhou, Elaine Shi, T-H. Hubert Chan, and Shir Maimon3. A theory of composition for differential obliviousness. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[ZZW24] Yulin Zhao, Hualin Zhou, and Zhiguo Wan. Superfl: Privacy-preserving federated learning with efficiency and robustness. *Cryptology ePrint Archive*, 2024.