UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Making Federated Analytics With Strong Privacy Practical

Permalink

https://escholarship.org/uc/item/679912h4

Author

Liu, Kunlong

Publication Date

2024

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California
Santa Barbara

Making Federated Analytics With Strong Privacy Practical

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Kunlong Liu

Committee in charge:

Professor Trinabh Gupta, Chair
Professor Amr El Abbadi
Professor Tao Yang

December 2024

The Dissertation of Kunlong Liu is approved.

Professor Amr El Abbadi

Professor Tao Yang

Professor Trinabh Gupta, Committee Chair

November 2024

Making Federated Analytics With Strong Privacy Practical

Copyright © 2024

by

Kunlong Liu

Acknowledgements

I thank my amazing advisor Trinabh Gupta who has guided me during the past five years. I've learnt a lot not only about research progress and technical writing but also commitment to doing solid work. Without the continuous support and endless efforts of my advisor, the past projects and this thesis would not be possible, especially during tough time. I also thank Natacha Crooks, Prabhanjan Ananth and Vivek Rudrapatna for their guidance in some projects. I also thank Amr El Abbadi and Tao Yang for their guidance, as well as serving on my committee.

I'm grateful for the collaborators I've had the fortune of working with during the past few years. Thank you for all of your hard work and commitment. I'm also grateful for my hosts during my summer internship at Google who helped expand my horizons. The process of doctoral studies is more than just an intellectual pursuit. I'm also grateful to have made friends in the UC Santa Barbara community, especially my lab mates and roommates. We've spent so much time during the past years together and helped each other through highs and lows of the PhD experience.

Finally, I want to thank my family for their dedicated and sustained support. They are the ones who have my back during the toughest time and I cherish so much their support.

Without all of you, I would not have reached this point. Thank you.

Curriculum Vitæ
Kunlong Liu

Education

2019 - 2024	Ph.D. in Computer Science (Expected), University of California, Santa Barbara, United States.
2014 - 2018	B.Sc. in Computer Science, Nanjing University, Nanjing, China.

Publications

Making Privacy-preserving Federated Graph Analytics Practical (for Certain Queries)
Kunlong Liu, Trinabh Gupta
ACM Symposium on Access Control Models and Technologies (SACMAT), 2024

Virtual Pooling: A method for privacy-preserving statistical analysis and machine learning over distributed networks of healthcare data
Ishtiyaque Ahmad, Kunlong Liu, Trinabh Gupta
UC Health Conference, 2024

Federated Learning with Differential Privacy and an Untrusted Aggregator
Kunlong Liu, Trinabh Gupta
International Conference on Information Systems Security and Privacy (ICISSP), 2024

Towards an efficient system for differentially-private cross-device federated learning
Kunlong Liu, Richa Wadaskar, and Trinabh Gupta
Workshop on Systems Challenges in Reliable and Secure Federated Learning (co-located with ACM SOSP) ResilientFL, 2021

Timestamped State Sharing for Stream Analytics
Yunjian Zhao, Zhi Liu, Yidi Wu, Guanxian Jiang, James Cheng, Kunlong Liu, and Xiao Yan
IEEE Transactions on Parallel Distributed Systems (TPDS), 2021

Accelerating 2PC-based ML with Limited Trusted Hardware
Muqsit Nawaz, Aditya Gulati, Kunlong Liu, Vishwajeet Agrawal, Prabhanjan Ananth, and Trinabh Gupta
arXiv, 2020

Abstract

Making Federated Analytics With Strong Privacy Practical

by

Kunlong Liu

Federated analytics, a way to perform analysis without collecting the raw data in a central place, has many applications nowadays. For example, Google trains next-word prediction models from mobile devices without collecting users' typing activities. Compared with traditional centralized approaches, federated analytics is preferred because federated analytics doesn't require devices to share their raw data with some central server since sharing raw data can be risky. The central server might abuse the raw data and there are chances of data breaches. However, achieving rigorous privacy guarantees for federated analytics can still be challenging due to the tension between functionality, privacy and efficiency. For example, one straw man solution is to apply general-purpose secure multiparty computation techniques, which meets the functionality and privacy goals but is expensive or even impractical for some queries.

A research question is, can we build federated systems that provide significant functionalities, rigorous privacy guarantees and are efficient at the same time? Our key insight is that there are ways to tailor the protocol and the system design for target applications such that the costs become practical and efficient while functionality and privacy guarantees are still provided.

As a step to answer this research question, we design and build three federated systems that target at different types of real queries, including machine learning tasks, graph queries and statistical queries from health literature. For both machine learning tasks and graph queries, we build systems that guarantee privacy without trusted cores, scale to millions of

devices, and have a low cost compared to previous works. For statistical queries, we build systems for tens or hundreds of health systems and provide same accuracy as centralized baseline while ensuring privacy for complicated statistical queries.

Contents

Chapter 1

Introduction

The rapid growth of personal data coupled with impressive advances in data science have revolutionized the world in many aspects. For example, companies use the data to build powerful tools to provide people with more convenient life and biomedical researchers learn from the data to gain fundamental insights into the basis of human health and disease and to make better decisions about disease treatment and prevention.

However, personal data can be extremely sensitive and sharing with analysts has significant privacy concerns. For example, in the scenario an analyst wants to train a recurrent neural network (RNN) to provide auto-completion suggestions for the android keyboard [1], devices need to share their typing history with the analyst, which may contain sensitive information including password, addresses and etc. Similarly, biomedical researchers may want to learn about some disease [2] and need health systems to share electronic health records data, which contains sensitive information such as age, ethnicity, addresses, smoking history and other medical information.

The traditional approach, where the analyst collects the data in a central repository, cannot address the increasing privacy concerns of the shared data. Firstly, this approach requires users to trust the analyst, otherwise users will not share the data. In the auto-completion sug-

gestions for the android keyboard [1] case, mobile users need to trust that the company will not abuse. In the biomedical case, the analyst needs to sign contracts with different health systems and this process can take months to years and some health systems may not be willing to sign contracts at all. Secondly, centralizing the data is highly susceptible to data breaches, especially in bulk [3, 4, 5, 6] and thus sharing raw data with analysts increases the risk of compromising users' data.

Federated Analytics is one diagram to perform analysis on distributed data that is owned by multiple parties under the coordination of a central entity without the raw data leaving local parties. The term was first coined by Google in 2020 [7] and one popular application of federated analytics is federated learning (FL). The core principle of federated analytics is to keep the raw data stored locally on users' devices, and only making the aggregated results across many devices available.

At first glance, federated analytics looks privacy-preserving since users' raw data will not leave users' devices. However, there are many possible attacks. Firstly, even users don't share the raw data, they still need to share some intermediate results with the analyst. For example, in federated learning, one common diagram requires each user to share its local model for aggregation, which has been shown to be vulnerable to inference attacks [8, 9, 10]. Attackers can reverse engineer to learn about or even reconstruct the raw data from the shared local model. Second, federated analytics assume that the analyst or the central server and the other users are honest. That is, these parties will not deviate from the protocol and try to learn about the data by compromising the system. In the federated learning case, when aggregating local models from users, a malicious server can manifest one local model by only including the victim's model in the aggregation. Similarly, compromised users can also provide malicious inputs to learn about the victim's data.

To guarantee privacy, many cryptographic based solutions are proposed. One approach is to apply local differential privacy (LDP). In LDP, devices *locally* add noise to their updates

before submitting them for aggregation [11, 12, 13, 14, 15, 16]. On the plus side, the privacy guarantee in LDP does not depend on the behavior of the aggregator, as devices add noises locally. Further, LDP is scalable as it adds small device-side overhead. However, on the negative side, since each device perturbs its update, the final results can have a large error.

A similar approach in statistical queries is distributed approximation [17, 18]. Since computing the exact results require devices to share the raw data, some works propose methods to approximate the results that can be computed in a way that only summary-level information will be leaked. This only applies to the setting where each party holds data for many individuals, e.g., health systems containing health records of many patients. This approach adds little overhead and the privacy does not depend on the behavior of the aggregator. However, the final results can have a large error if the underlying assumptions are not met. For example, different data distribution across health systems and not enough number of samples.

An alternative is to use trusted hardware, e.g., Intel SGX [19]. Secure hardware ensures by hardware design that data and computation within the secure hardware cannot be accessed by anyone else. These systems add negligible overhead and introduce no errors, but trusting the hardware design and manufacturer is a strong assumption [20, 21, 22].

To have good accuracy without relying on trust of parties, many works use cryptographic tools, such as secure multi-party computation (MPC) and homomorphic encryption (HE) [23, 24, 25, 26, 27]. At a high level, users encrypt their updates before submitting to the aggregator. The aggregator performs computation over the encrypted data and asks users to decrypt the results before releasing. Works in this category can be further divided by their threat model, e.g., whether the aggregator is honest-but-curious or can be active malicious. Crypto-based solutions usually have good accuracy since it does the exact computation as the traditional centralized approach but in an encrypted fashion. The down side is the performance, especially for a strong threat model where parties can be malicious. Some existing works show that some federated analytics systems are super expensive or even impractical [28, 29, 30].

In this dissertation, we focus on methods using cryptographic tools to achieve privacy-preserving federated analytics due to its advantages of good accuracy and no trust on single parties. The research question we ask is, *Can we build federated systems that provide significant functionality, provide rigorous privacy guarantees, and are efficient at same time?*

Federated analytics is a common diagram and can be used to answer many types of queries. As a first step to answer this question, we pick three different real-world queries, namely machine learning tasks (Chapter 3), graph queries (Chapter 4) and statistical queries (Chapter 5). They differ from each other in several aspects and we will detail it in the later chapters.

Machine learning tasks, or federated learning, has seen many applications in industry [1, 31, 32, 33]. In FL, mobile devices download the latest model parameters from the server, perform *local* training to generate *updates* to the parameters, and send only the updates to the server. The server takes the average of the updates and update the global model for next round of training. Graph queries have also seen many applications [34, 35] and they differ from federated learning in how updates are computed. In FL, the computation of local updates are independent from other users but for graph queries, the computation of local updates requires inputs from other users (neighbors), which leads to a different and more complicated computational model than FL and brings more privacy risks. Statistical queries are motivated from real world analysis across multiple health systems [2] and differ from previous two queries in the global aggregation part. In FL and graph queries, the aggregation operation is simple averaging. But in statistical queries, the aggregation operations are usually much more complicated. The takeaway here is these three types of queries are all important in real world and they represent three different types of applications.

1.1 Challenges and Goals

The challenge of building practical federated analytics systems lies in addressing the tension between functionality, privacy and efficiency. Existing works usually achieve two of the three goals and fail to meet the other one. For example, LDP-based works meet privacy and efficiency goals but fail to meet our functionality goal due to the accuracy issues. Works assuming not strong enough adversaries can meet the functionality and efficiency goal but not the privacy goal. Instead, works assuming no trusted cores such as Orchard [28] fail to meet the efficiency goal but meets the other two goals. For our target queries, we have not yet seen systems that can satisfies these three goals at the same time.

Goal 1: Functionality. We want the system to support our target queries but we don't aim to be general-purpose. For example, in FL, we focus on a specific training algorithm called FedAvg [36]. In graph queries, we are also limited to a certain set of queries. This is because previous works have shown that a general-purpose system with strong privacy are super expensive and impractical [28, 30]. However, we do aim to provide results as if the data were centralized and the analysis was done over the centralized data. The intuition behind is the loss of accuracy for existing methods is usually hard to quantify, which means the system might not produce reliable results in some scenarios.

Goal 2: Strong privacy guarantees. We want the system to provide rigorous and strong privacy guarantees. For different scenarios, we have different threat models. However, the threat models must be practical. For example, in FL, since our target setting is millions of devices and a central server, we would like the system to tolerate compromise of a fraction of devices and compromise of the server, since it's easier for an adversary to compromise some devices and deviate from the protocol and it's also possible for some attackers to compromise the company servers. However, in a setting where only tens of hospitals participate in the system, it's unlikely that a hospital will deviate from the protocol.

Goal 3: Efficiency. Of course the system overhead should be low. For different scenarios, we focus on different metrics. In FL and graph queries, since mobile devices have limited computational power and network bandwidth, we would like the device-side cost to be as low as possible and the server-side cost to be practical. Instead, for the statistical queries, since data owners are hospitals, we focus more on bandwidth and number of communications rounds.

1.2 Approach

There are a few insights we can make about building a practical federated analytics system with strong privacy guarantees. First, we can decouple privacy guarantees with correctness guarantees and focus on our primary goal — privacy. Let's use a toy example in federated learning to illustrate this. Recall that devices need to upload their local models for aggregation. Consider the following attack: a malicious device uploading a model with all parameters set to be 0. This attacks definitely compromise correctness, since the global model will differ from a model trained with the centralized baseline. However, privacy will not be compromised since privacy guarantees don't rely on what local update is and it's possible that some local update consists of all 0. So a short conclusion is, privacy and correctness are two different guarantees and in some cases, privacy is a "weaker" guarantee than correctness. Another observation is, it's almost impossible for us to ensure correctness if some devices are malicious. Still for the toy example, if the server receives some local update, how can it tell whether it's "correct" or not? It's hard to tell because a malicious device can start from some desired local update and construct a dataset that will output the desired update. So the first insight is we can tailor the protocol and system design for a finer-grained threat model, e.g., privacy guaranteed when devices and the server are malicious and correctness guaranteed only when all devices are honest-but-curious.

The second insight we make is we can tailor the system design and protocol for our target queries. As discussed before, building a general-purpose system with strong privacy is usually impractical. So instead of being general-purpose, we want our systems to support a set of queries and thus we can find characteristics of the workload and tune the system for better efficiency. For example, in FL, we focus on a specific type of training algorithm called FedAvg [36] and one characteristics of this algorithm is sampling. That is, only a small subset of devices will be sampled for contributing local updates in each round. We build a system that uses this characteristic for better performance. We also apply this idea in graph queries and statistical queries.

The third insight we make is to have an uniform privacy guarantees across different system components. This is important especially when the system is large and complicated. For example, in FL, we need to securely sample devices, aggregate updates, generate noise and decrypt the ciphertexts, and etc. The system has many components due to the complicated workload and each component can be addressed using different MPC protocols. A lesson we've learned from existing works and apply to our system design is to tailor the protocols for each component to make them have same privacy guarantees. The reason behind is straightforward: the system's privacy or correctness guarantee is decided by the component with weakest privacy or correctness guarantees. Any components with stronger privacy or correctness guarantees incur unnecessary overhead.

In short, we tune system architecture and protocols to our target workload to gain performance improvement while ensuring that we can answer target queries with strong and rigorous privacy guarantees, and thus to improve the trade-off between functionality, privacy and efficiency,.

1.3 Contributions

Our novel contributions in this dissertation are as follows:

- Aero, previously published as [37, 38, 39], a new federated learning system that guarantees good accuracy, differential privacy over an untrusted server, and keeps the device overhead low. An evaluation of Aero demonstrates that it provides comparable accuracy to plain federated learning (without differential privacy), and it improves efficiency (CPU and network) over state-of-the-art Orchard [28] by up to $10^5\times$.

- Colo, previously published as [40], a new and low-cost system for privacy-preserving federated graph analytics for a particular subset of queries. An implementation and evaluation of Colo shows that for running a variety of COVID-19 queries over a population of 1M devices, it requires less than 8.4 minutes of a device's CPU time and 4.93 MiBs in network transfers—improvements over state-of-the-art Mycelium [30] of up to three orders of magnitude.

- Virtual Pooling, previously published as [41], which is the first system for federated statistical analytics that produces the same results as the centralized baseline and supports a set of statistical methods including descriptive statistics, propensity score matching, Cox Proportional hazard models. An implementation and evaluation of Virtual Pooling shows that it can produce exactly the same results for an end-to-end analysis pipeline and it incurs an additional latency of 17 s for a centralized baseline that takes about 5 min.

Chapter 2

Background

In this chapter, we give an overview of some cryptographic tools we've used in building our privacy-preserving federated systems.

2.1 Differential Privacy (DP)

Differential privacy [42] is a mathematical framework that allows for meaningful data analysis while protecting individual privacy. The core idea is that the presence or absence of any single individual's data in a dataset should not significantly affect the outcome of any analysis. In other words, differential privacy ensures that any insights derived from the data are general enough that an observer cannot ascertain whether any specific individual's data is included. This guarantee is quantified in the form of two parameters, ϵ, δ, which describes how much a single row can change the results.

Formally, let D_1 and D_2 be any two neighboring datasets that differ only in a single row, $\mathcal{A}$ be a randomized algorithm. The algorithm $\mathcal{A}$ is said to provide (ϵ, δ)-differential privacy if, for all neighboring datasets D_1 and D_2, and all subsets $\mathcal{S}$ of the image of $\mathcal{A}$:

$$Pr[\mathcal{A}(D_1) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathcal{A}(D_2) \in \mathcal{S}] + \delta$$

In other words, a change of single row has a bounded probability to cause a change in the output. Differential privacy is widely used in machine learning to defend against inference attacks [8, 9, 10]. The intuition behind is, assuming the real input is D_1 and given the output, the adversary cannot tell whether the input is D_1 or any neighboring dataset of D_1. The smaller ϵ, δ are, the smaller probabilities to distinguish between neighboring datasets and thus strong privacy guarantees.

One way to achieve differential privacy is to add noise to data or query results to protect individual privacy, making it difficult to infer specific information about any single data point. One common method for achieving this is through the addition of Gaussian noise.

The amount of noise added depends on ϵ, δ, as well as the sensitivity of the query, which refers to the maximum amount that a single individual's data can change the outcome of a query. The Gaussian noise added to each data point or query result is scaled based on the sensitivity of the query and the desired privacy level (ϵ, δ). However, this also means a potential trade-off in data accuracy, as higher noise can obscure finer details of the dataset. So ideally we would like ϵ, δ to be as small as possible for stronger privacy guarantees, but in practice, we follow Google [32] to set ϵ near 5.

Sampling can be combined with noise mechanisms like Gaussian to amplify privacy guarantees. One commonly used approach is randomized sampling, where a random subset of data points is selected from the dataset before applying a privacy-preserving mechanism. This randomized sampling step reduces the chance that any individual's data will be included in a given sample, effectively adding an additional layer of protection. For example, in machine learning, models can be trained on random mini-batches of data, where each mini-batch is treated as a sample, allowing the model to learn patterns while preserving the privacy of indi-

viduals in the larger dataset. This approach ensures that privacy loss is spread out over many iterations, providing cumulative protection throughout the model training process.

However, sampling doesn't work well in privacy-preserving federated learning. There are two reasons. First, it's hard to do uniform sampling since devices can go online and offline during the training. Some devices might participate more in the training and some devices might never participate. Second, usually we assume the adversary can observe the traffic and thus be able to tell which devices talk to the server. That is, the adversary learns which devices contribute to the training. One key idea of sampling amplification is that the adversary only knows each device has a probability to participate but doesn't know which rounds each device participate in.

2.2 Secure Multi-Party Computation (MPC)

Secure multi-party computation (MPC) is a cryptographic framework that allows multiple parties to collaboratively compute a function over their inputs while keeping those inputs private. MPC has been researched extensively by cryptographers and security researchers for several decades and there are many protocols and primitives. So in this chapter, we only give an overview of two techniques we've used in our systems.

One technique we've used in our systems is the Shamir's secret sharing scheme [43], where each input is divided into "shares" that are distributed across parties so that no single party has complete information. Two primitives of Shamir secret sharing are additions and multiplications. Parties work together to perform computation over the shared data and get results in a shared manner. At the end of the protocol, parties collaborate to reconstruct the results. During the computation, only the final results will be revealed. One key characteristic of Shamir secret sharing is that reconstructing a secret requires a threshold of parties. For example, if a secret is shared among 5 parties, with Shamir secret sharing, we can ensure that at

least 3 parties are required to reconstruct the secret. Of course, this threshold is configurable.

Here we give a toy example. Suppose there are 3 parties S_1, S_2, S_3 and they have a, b, c as private inputs. The goal is to compute $a \cdot b \cdot c$ without revealing any information about a, b, c.

- S_1 generates shares of a, say, $a_1, a_2, a_3 = Share(a)$ and send $a_2 \to S_2$, $a_3 \to S_3$. And we define $[a]$ to be a in its sharing form.
- S_2 generates shares of b, say, $b_1, b_2, b_3 = Share(b)$ and send $b_1 \to S_1$, $b_3 \to S_3$. And we define $[b]$ to be b in its sharing form.
- S_3 generates shares of c, say, $c_1, c_2, c_3 = Share(c)$ and send $c_1 \to S_1$, $c_2 \to S_2$. And we define $[c]$ to be c in its sharing form.
- S_1, S_2, S_3 run multiplication protocol to compute $[a] \cdot [b] \cdot [c]$ to get the result, say $[y]$.
- S_1, S_2, S_3 run reconstructing protocol to reconstruct y as the public output (result).

Another technique we've used in our systems is the Oblivious Transfer (OT) [44], which is a cryptographic protocol that enables a sender to transfer information to a receiver in a way that protects the privacy of both parties. Specifically, the sender has multiple pieces of information but remains unaware of which piece the receiver has chosen, while the receiver learns only the selected information without gaining access to the other options.

The most commonly used form of OT is 1-out-of-2 Oblivious Transfer, where the sender has two messages, M_0 and M_1, and the receiver wants to select one of these messages, say M_b, where $b \in \{0, 1\}$. The protocol ensures that:

- The sender does not learn which message the receiver selected.
- The receiver learns only the selected message M_b and gains no information about the other message.

2.3 Homomorphic Encryption (HE)

Homomorphic Encryption (HE) is an advanced cryptographic technique that enables computations to be performed directly on encrypted data without needing to decrypt it. In our systems, we are focused on RLW-based encryption schemes such as BFV [45] and BGV [46].

HE schemes consists one pair of keys: public key pk and secret key sk. The public key is used to encrypt the data into ciphertexts and can be known to everyone. The private key is used to decrypt the data and thus should be kept private.

Formally, HE schemes have Keygen, Encrypt, Decrypt functions:

- $pk, sk \leftarrow Keygen(1^{\lambda})$, where λ is the security parameter.
- $ct = Enc(pk, m)$, where m is the message.
- $m = Dec(sk, ct)$, where ct is the ciphertext.

In HE, adding or multiplying ciphertexts translates to adding or multiplying the encrypted messages. For example, if two messages are m_1, m_2 and their ciphertexts are $ct_1 = Enc(pk, m_1)$ and $ct_2 = Enc(pk, m_2)$, then

- $m_1 + m_2 = Dec(sk, ct_1 + ct_2)$
- $m_1 * m_2 = Dec(sk, ct_1 * ct_2)$

In RLWE-based schemes, e.g., BFV, messages are usually encoded into polynomials for encryption. So one ciphertext can carry multiple numbers, which are also called slots. One example HE scheme enables us to encrypt 4096 numbers in one ciphertexts and we can add or multiply these 4096 numbers at the same time.

One challenge with HE is its computational cost, as encrypted operations are significantly slower than regular operations due to the added complexity of managing encrypted data.

There are many optimizations, including Number Theoretic Transform (NTT) and Chinese Remainder Theorem (CRT) encoding. However, the number of multiplication operations, or specifically multiplication depths can make the costs high. There are two reasons. Firstly, HE is encoded over some field, e.g., a 108-bit prime number. Increasing multiplications usually require us to switch to a larger field, which makes each ciphertext larger and also computational cost higher. Second, HE adds noise during encryption and multiplications increase the noise scale very fast and thus at some point we need some expensive operations such as modulus switching or relinearization.

2.4 Zero-Knowledge Proofs (ZKP)

Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that allow one party, known as the prover, to demonstrate to another party, the verifier, that a certain statement is true without revealing any additional information beyond the validity of the statement itself. In our system, we use Groth16 [47].

Here we give a toy example to illustrate how ZKP works. Suppose in FL, a device wants to prove to the server that ct is a valid encryption under the public key pk.

- Setup: the server needs to generate a pair of keys $ProvingKey$ and $VerifyKey$. The $ProvingKey$ is bound to a circuit, which is $y = Enc(pk, x)$, where x is the private secret and y is the public output.
- The device receives $ProvingKey$ and uses $y = ct$ and $x = m$ to generate a proof, where ct is the ciphertext and m is the message.
- The device sends ct along with the proof and the server uses the $VerifyKey$ to verifies whether the device really knows a m that satisfies $ct = Enc(pk, m)$.

There are some key benefits of Groth16.

- Constant-Size Proofs: The proof size is constant (approximately 192 bytes), regardless of the complexity of the computation.
- Fast Verification: the verification is fast, usually in terms of milliseconds.
- Non-Interactive: Groth16 is a non-interactive proof, meaning that the prover generates a proof that the verifier can independently verify without the need for back-and-forth communication.

However, these benefits come at the cost of proving time. If the circuit to prove is large, it can take a long time to prove. In our toy example of proving validity of encryption, one proving can cost tens of minutes CPU time.

Chapter 3

Aero

In this chapter, we describe Aero, which significantly improves the trade-off between accuracy, privacy, and device overhead for our first target type of queries — federated learning tasks.

Federated learning (FL) is a recent paradigm in machine learning that embraces a decentralized training architecture [48]. In contrast to the traditional, central model of learning where users *ship* their training data to a central server, users in FL download the latest model parameters from the server, perform *local* training to generate *updates* to the parameters, and send only the updates to the server. Federated learning has gained popularity in training models for mobiles [1, 31, 32, 33] as it can save network bandwidth and it is privacy-friendly—raw data stays at the devices.

Current systems for federated learning exhibit significant trade-offs between model accuracy, privacy, and device efficiency. For instance, one class of systems that includes Oort [49], FedScale [50], and FedML [51] provides excellent accuracy (comparable to centralized learning) and device efficiency. But these systems provide only a weak notion of privacy. This point is subtle. At first glimpse, it appears that in any federated learning system, since users ship updates to model parameters rather than the raw training data, this data (user images, text

messages, search queries, etc.) remains confidential. However, research shows that updates can be reverse-engineered to reveal the raw data [8, 9, 10]. Thus, if the server is compromised, so is the users' data. In other words, the server must be trusted.

On the other hand, systems such as HybridAlpha [52] and Orchard [28] offer good accuracy and a differential privacy guarantee for users' data. Informally, differential privacy says that an adversary cannot deduce a user's training data by inspecting the updates or the learned model parameters [53, 42, 54, 55]. In fact, Orchard guarantees differential privacy while assuming a byzantine server. But the downside is the high overhead for the devices. For example, to train a CNN model with 1.2 million parameters [56], Orchard requires from each device ≈14 minutes of training time on a six-core processor and ≈840 MiB in network transfers per round of training (§3.5.2). The full training requires at least a few hundred rounds. Further, for a few randomly chosen devices, this per-round cost spikes to ≈214 hours of CPU time and ≈11 TiB of network transfers. Clearly, this is quite high.[1]

Aero provides good accuracy, the differential privacy guarantee in the same threat model as Orchard, and low device overhead. For instance, most of the time Aero's devices incur overhead in milliseconds of CPU time and KiBs of network transfers.

The key idea in Aero is that it does not aim to be a general-purpose federated learning system, rather focuses on a particular class of algorithms (§3.2.1). These algorithms sample devices that contribute updates in a round using a simple probability parameter (e.g., a device is selected with a probability of 10^{-5}), then aggregate updates across devices by averaging them, and generate noise needed for differential privacy from a Gaussian distribution. Admittedly, this is only one class of algorithms, but this class comprises popular algorithms such as DP-FedAvg [36] and DP-FedSGD [36] that are the ones commonly used and deployed [1, 31, 50, 51]. With this restriction, Aero tunes system architecture and design

[1] Another class of systems provides a particular type of differential privacy called local differential privacy (LDP) [11, 13, 16]. These systems are efficient but LDP creates a high accuracy loss [16, 57, 13] (§3.1.3, §3.6).

to these algorithms, thereby gaining on performance by orders of magnitude.

This tuning is non-trivial and requires novel techniques and optimizations (§3.2.2, §3.3). As one example, the devices must verify that the byzantine server did not behave maliciously. One prior technique is to use a summation tree [29], where the server explicitly shows its work aggregating updates across devices in a tree form; the devices then collectively check nodes of this tree. This checking, in turn, adds overhead to the devices. Aero addresses this tension between privacy and overhead by leveraging the sampling characteristic of DP-FedAvg and similar algorithms: the total number of devices that participate in the system (e.g., one billion) is much larger than the ones that are sampled to contribute updates. Leveraging this characteristic, Aero employs multiple, finer-grained summation trees (rather than a monolithic tree) to massively divide the checking work across the large device pool (§3.3.3). Aero further optimizes how each device verifies the nodes of the tree using a technique called polynomial identity testing (§3.3.3). The aforementioned is just one example of optimization; Aero uses multiple throughout its architecture (§3.2.2) and design (§3.3).

We implemented Aero by extending the FedScale FL system [50] (§3.4). FedScale supports plain federated learning without differential privacy or protection against a byzantine server. However, it is flexible, allows a programmer to specify models in the PyTorch framework [58], and includes a host of models and datasets, with model sizes ranging from 49K to 3.9M, for easy evaluation. Our evaluation of Aero's prototype (§3.5) shows that Aero trains models with comparable accuracy to FedScale, in particular, the plain FedAvg algorithm in FedScale (§3.5.1). Aero also improves overhead relative to Orchard by up to five orders of magnitude, to a point where the overhead is low to moderate. For instance, for a 1.2M parameter CNN on the FEMNIST dataset [56, 59], and for a total population of 10^9 devices where 10^4 contribute updates per round, a Aero device requires 15 ms of CPU time and 3.12 KiB of network transfers per round. Occasionally (with a probability of 10^{-5} in a round) this overhead increases when a device contributes updates, to a moderate 13.4 minutes of latency on a six-core processor

and 234 MiB in network transfers.

Prior to Aero, one must choose two of the three properties of high accuracy, rigorous privacy guarantee, and low device overhead. With Aero, one can train models in a federated manner with a balance across these properties, at least for a particular class of federated learning algorithms. Thus, Aero's main contribution is that it finally shows a way for a data analyst to train models while asking the data providers to place no trust in the analyst or their company.

3.1 Problem and background

This section outlines the problem and gives a short background on Orchard [28] that forms both a baseline and an inspiration for Aero.

3.1.1 Scenario and threat model

We consider a scenario consisting of a data analyst and a large number of mobile devices, e.g., hundreds of million. The analyst, perhaps at a large company such as Google, is interested in learning a machine learning model over the data on the devices. For instance, the analyst may want to train a recurrent neural network (RNN) to provide auto-completion suggestions for the android keyboard [1].

One restriction we place on this scenario is that the training must be done in a federated manner. (We refer the reader to prior work [60, 61] for a discussion on training in the centralized, non-federated model.) As noted earlier, federated learning proceeds in rounds, where in each round devices download the latest model parameters from a server, generate updates to these parameters locally, and send the updates to the server. The server aggregates the model updates. This repeats until the model achieves a target accuracy.

In this scenario, a malicious server, or even a malicious device, can execute many attacks.

For instance, a malicious server can infer the training data of a device from the updates contributed by the device [8, 9, 10]. Similarly, a malicious device that receives model parameters from the server can execute an inference attack to learn another device's input.

We assume the strong *OB+MC* threat model from Orchard. The server is honest-but-curious most of the time but occasionally byzantine (OB), while the devices are mostly correct (MC), but a small fraction can be malicious. The rationale behind the occasionality of the server's maliciousness is that the server's operator, e.g., Google, is reputed and subject to significant scrutiny from the press and the users, and thus unlikely to be byzantine for long. However, it may occasionally come under attack, e.g., from a rogue employee. The rationale behind the smallness of the fraction of malicious devices is that with billions of devices, it is unlikely that an adversary will control more than a small percentage. For instance, for a billion devices, only controlling 3% is already significantly larger than a large botnet. We further assume that a configurable percentage of honest devices may be offline during any given round of training.

3.1.2 Goals

Under the OB+MC threat model, we want our system to meet the following goals.

Privacy. It must guarantee the gold standard definition of privacy, i.e., differential privacy (DP) [53, 42, 54, 55]. Informally, a system offers DP for model training if the probability of learning a particular set of model parameters is (approximately) independent of whether a device's input is included in the training. This means that DP prevents inference attacks [62] where a particular device's input is revealed, as models are (approximately) independent of its input.

Accuracy. During periods when the server or the devices that contribute in a round are not byzantine, our system must produce models with accuracy comparable to models trained via

plain federated learning. That is, we want the impact of differential privacy to be low. Further, we want the system to mitigate a malicious device's impact on accuracy and prevent it from supplying arbitrary updates.

Efficiency and scalability. We want the system to support models with a large number of parameters while imposing a low to moderate device-side overhead. For the former, a reference point is the android keyboard auto-completion model (an RNN) with 1.4M parameters [1]. For the device overhead, if a device participates regularly in training, e.g., in every round, then it should incur no more than a few seconds of CPU and a few MiBs in network transfers per round. However, we assume that devices can tolerate occasional amounts of additional work, contributing tens of minutes of CPU and a few hundred MiBs in network transfers.

3.1.3 Possible solution approaches

Meeting all of the goals described above is quite challenging. For illustration, consider the following solution approaches.

Local differential privacy. One option to guarantee differential privacy is to pick a federated learning algorithm that incorporates local differential privacy (LDP) [11]. In LDP-based federated learning, devices add statistical noise to their updates before uploading them to the server. The added noise protects updates against a malicious server which now cannot execute an inference attack, but LDP significantly degrades model accuracy relative to plain federated learning as every device must add noise. For instance, the LDP-FL [13] system trains a VGG model over CIFAR10 with 10% accuracy compared to 62% with plain federated learning.

Trusted server. One alternative is to use central differential privacy [55], where a central entity adds a smaller amount of noise to device updates to ensure differential privacy. This

approach mitigates the accuracy issue, but provides weak to no privacy as the central entity sees devices' updates.

Server-side secure multiparty computation (MPC). One way to reduce trust in the central server is to break it down into multiple non-colluding pieces, e.g., three servers, that run in separate administrative domains. Then, one would run a secure multi-party computation protocol [63, 64] among these servers such that they holistically perform the required computation (noise generation, addition, etc.) while no individual server sees the input or intermediate state of the computation. The problem is that we must still put significant trust in the server—that an adversary cannot compromise, say, two administrative domains.

Large-scale MPC. One can remove trust in the server by instead running MPC among the devices (essentially the devices perform the server's work). The problem now shifts to efficiency and scalability: general-purpose MPC protocols are expensive and do not scale well with the number of participants [65, 66]. Indeed, scaling MPC to a few hundred or thousand participants is an active research area [67, 68], let alone hundreds of millions of participants.

State-of-the-art: Orchard. Orchard [28] takes a middle ground. It runs small(er)-scale MPC among devices while assuming an (occasionally) byzantine server. In particular, it forms a *committee* of a few tens of devices picked randomly from the entire population of devices; this committee then runs MPC among its members. Figure 3.1 shows an overview of Orchard. Orchard supports the full-batch gradient descent algorithm where all devices contribute updates in a round. For every round, Orchard runs the following four phases.

In the setup phase, Orchard samples the committee. Its members use MPC to generate keys for two cryptographic primitives: additive homomorphic encryption (AHE) [69, 45] and zero-knowledge proof (ZK-proof) [47, 70].

In the generate phase, devices download the latest model parameters from the server. After local training, they encrypt their model updates (denoted by Δ in Figure 3.1) and generate

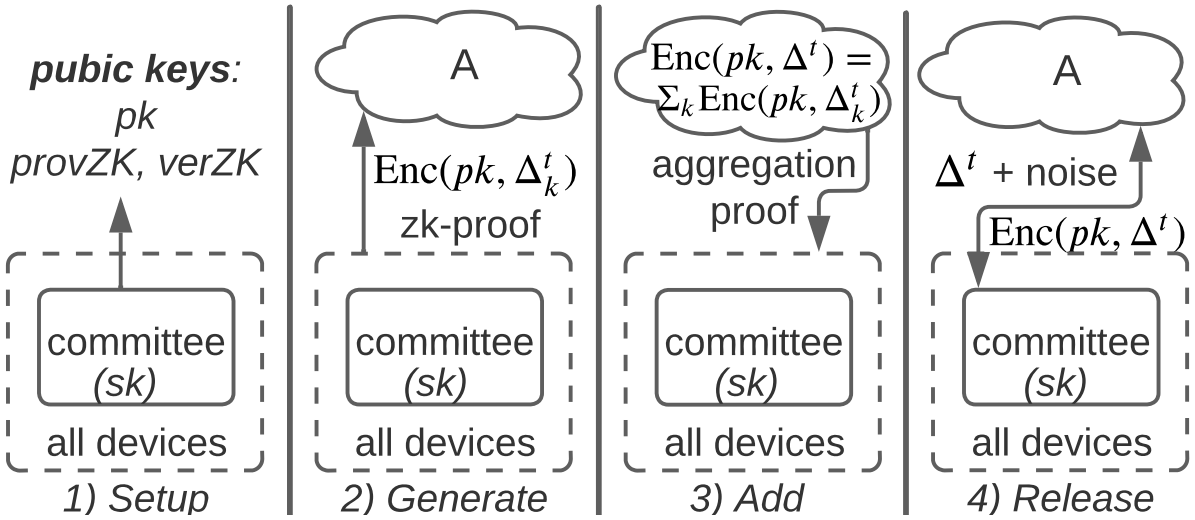


Figure 3.1: An overview of Orchard [28]. Δ_k denotes k-th device's update. The superscript t denotes the round number. Orchard runs the four phases of setup, generate, add, and release for every round.

a ZK-proof to prove to the server that the ciphertexts are well-formed and the model updates are bounded. The encryption hides the updates from the byzantine server and the proof limits the impact of malicious devices (they cannot supply arbitrary updates and thus ruin the model accuracy).

In the add phase, the server homomorphically adds the encrypted updates. The server also generates proof that it performed the addition correctly so that all devices can collaboratively verify the addition. This verification is necessary to prevent a byzantine server from launching subtle attacks to break DP. (We will discuss these attacks further in §3.3.3.)

In the release phase, the committee from the setup phase uses MPC to decrypt the output ciphertexts from the add phase. The committee also generates and adds the DP noise to the output, before releasing it, to guarantee (central) DP.

The challenge with Orchard is that even though it uses MPC at a smaller scale, the MPC overhead is still high. First, in the setup phase, committee devices generate fresh keys for each round, and generating one AHE key pair inside general-purpose MPC requires ≈180 seconds of CPU and 1 GiB of network transfers. Second, the overhead of the add phase to verify the

server's work grows linearly with the model size and becomes unpragmatic as soon as the model has a few hundred thousand parameters. Third, in the release phase, committee devices decrypt ciphertexts and generate DP noise inside general-purpose MPC, costing, for example, ≈2600 seconds of CPU and ≈38 GiB of network transfers per device for a model with just 4K parameters.

In general, providing high accuracy, differential privacy, and device efficiency simultaneously in a threat model where there is no trusted party proves incredibly challenging.

3.2 Overview of Aero

The high-level idea in Aero is to focus on a specific type of federated learning algorithms comprising DP-FedAvg [36], DP-FedSGD [36], and DP-FTRL [32]. These algorithms have similar characteristics; for instance, they all sample noise for differential privacy from a Gaussian distribution. To keep Aero easier to explain and understand, we take the most popular DP-FedAvg as the canonical algorithm and describe Aero in its context.

3.2.1 DP-FedAvg without amplification

DP-FedAvg proceeds in discrete rounds (Figure 3.2). In each round t, it samples a small subset of user devices using a probability parameter q (line 8), and asks the sampled devices to provide updates to the global model parameters (line 10). The devices locally generate the updates before clipping them by a value S and uploading them (line 21); this clipping is necessary for differential privacy and it bounds the norm (sensitivity) of a device's update. DP-FedAvg then aggregates these updates (line 12) and (separately) adds noise sampled from a Gaussian distribution. The standard deviation of the Gaussian distribution depends on a noise scale parameter z and the clipping bound S; both are input parameters for DP-FedAvg. Finally, DP-FedAvg updates a privacy accountant $\mathcal{M}$ that computes, based on the noise scale

MAIN:

1: *parameters*
2: device selection probability $q \in (0, 1]$
3: DP noise scale z
4: total # of devices W
5: clipping bound on device updates S
6: Initialize model θ^0, DP privacy budget accountant $\mathcal{M}$
7: **for** each round $t = 0, 1, 2, \ldots$ **do**
8: $C^t \leftarrow$ (sample users with probability q)
9: **for** each user $k \in C^t$ **do**
10: $\Delta_k^t \leftarrow \text{USERUPDATE}(k, \theta^t, S)$
11: // Add updates and Gaussian DP noise with $\sigma = zS$
12: $\Delta^t \leftarrow \sum_k \Delta_k^t + \mathcal{N}(0, I\sigma^2)$
13: $\theta^{t+1} \leftarrow \theta^t + (\Delta^t/(qW))$ // Update model
14: Update $\mathcal{M}$ based on noise scale z and parameter q

USERUPDATE(k, θ^0, S)

15: *parameters* B, E, η // η is learning rate
16: $\theta \leftarrow \theta^0$
17: **for** each local epoch i in 1 to E **do**
18: $\mathcal{B} \leftarrow$ (k's data split into size B batches)
19: **for** batch $b \in \mathcal{B}$ **do**
20: $\theta \leftarrow \theta - \eta \nabla \ell(\theta; b)$ // ℓ is loss fn. (model err.)
21: $\theta \leftarrow \theta^0 + Clip(\theta - \theta^0, S)$
22: **return** $\Delta_k = \theta - \theta^0$ // Already clipped

Figure 3.2: Pseudocode for the DP-FedAvg algorithm. $Clip(\cdot, S)$ scales its input vector such that its norm (Euclidean distance from the origin) is less than S. $\mathcal{M}$ is the privacy budget accountant of Abadi et al. [55] that tracks the values of the DP parameters ϵ and δ.

z and sampling probability q, two parameters ϵ and δ associated with differential privacy (line 14). These parameters capture the strength of the guarantee: how much the model parameters learned after a round vary depending on a device's input. A lower value of δ and ϵ is desirable, and the literature recommends ensuring that ϵ stays close to or below 1, and δ is less than $1/W$, where W is the total number of devices [36].

DP-FedAvg has three characteristics that are crucial for Aero. The first is the sampling of devices (lines 8 to 10 in Figure 3.2). For instance, the sampling parameter q could be 10^{-5} such that 10,000 out of, say, 10^9 total devices contribute updates in a round. The second characteristic is that the noise is sampled from a Gaussian distribution whose standard deviation σ

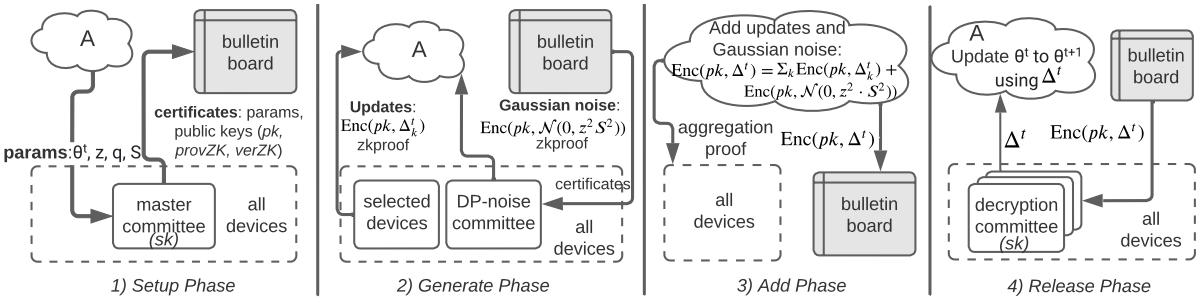


Figure 3.3: An overview of Aero's architecture and the four phases of its protocol.

is predetermined (set before the algorithm is run). This is in contrast to other DP algorithms that utilize techniques such as the Sparse Vector Technique (SVT) that generate noise depending on the value of the updates [71, 72]. The third characteristic is averaging of updates: DP-FedAvg simply adds updates and noise (line 12 in Figure 3.2) rather than combining them using a more complex function. Aero heavily leverages these characteristics.

Finally, we remark that Aero can support DP-FedAvg only without the amplification assumption for DP. This is because the adversary (the byzantine server) can observe all traffic and knows which devices contribute updates for training. In contrast, the amplification assumption requires the server to be oblivious to the contributors, which in turn improves the privacy budget. We leave the addition of expensive oblivious approaches (which hides who is contributing updates besides hiding the updates themselves) to future work.

3.2.2 Architecture of Aero

Aero borrows two system components from Orchard: an *aggregator* and a public *bulletin board* (Figure 3.3). The aggregator runs server-side inside a data center and therefore consists of one or more powerful machines. Its main role is to combine updates from user devices without learning their content. The bulletin board is an immutable append-only log. The aggregator (which is potentially malicious) and the devices use the bulletin board to reliably broadcast messages and store states across rounds, e.g., the latest values of DP parameters ϵ

and δ. Like Orchard [28], Aero assumes that free web services such as Wikipedia, or a public block-chain could serve as the bulletin board.

Like Orchard, Aero also consists of *committees* of devices, but instead of a single committee as in Orchard, Aero has three types of committees tailored to the needs of DP-FedAvg (and similar algorithms). A *master committee* handles system setup, including key generation for cryptographic primitives. A *DP-noise committee* handles Gaussian noise generation. And multiple *decryption committees* perform decryption operations to release updates to the global model parameters at the end of a training round. Aero samples each committee afresh each round, dividing the committee workload across the large population of devices.

An architecture with separate committee types is deliberate and crucial. It helps tailor a committee's protocol to its tasks to significantly improve efficiency. Besides, the use of multiple committees of the same type, i.e., multiple decryption committees, helps Aero scale with model size as each committee works on a subset of model parameters.

Notably, the ability to use separate committee types is possible only because of the specifics of DP-FedAvg. For instance, the fact that Gaussian noise generation does not depend on the value of the updates allows Aero to separate the DP-noise committee from the decryption committees.

3.2.3 Protocol overview of Aero

To begin training a model, a data analyst supplies input parameters (the model architecture, the initial model parameters, and the other input parameters for DP-FedAvg) to the aggregator. The aggregator then initiates a *round-based protocol* consisting of discrete rounds. In each round, it executes one iteration of the for loop in the Main procedure of DP-FedAvg (line 7 in Figure 3.2). Each round further consists of the four phases of *setup*, *generate*, *add*, and *release* (Figure 3.3).

In the setup phase, the aggregator samples the various committees for the round. The master committee then receives and validates the input parameters, and generates keys for an AHE and a ZK-proof scheme. Aero's setup phase is similar to Orchard (§3.1.3) with the key difference that Aero's master committee uses techniques to reuse keys across rounds rather than generating them fresh for each round using MPC.

Next, in the generate phase, (i) devices select themselves to generate updates for the round, and (ii) the DP-noise committee generates the Gaussian noise for DP. Both types of devices use techniques to perform their work efficiently. For instance, the DP-noise committee generates noise in a distributed manner while avoiding MPC.

Next, in the add phase, the aggregator adds the model updates to the Gaussian noise without learning the plaintext content of either of them. This is done through the use of the AHE scheme. The entire population of devices collectively verifies the aggregator's work. Again, the key is efficiency for the devices, for which the aggregator and the devices use a new verifiable aggregation protocol.

Finally, in the release phase, each decryption committee receives the secret key for the AHE scheme from the master committee and decrypts a few ciphertexts from the add phase. The key point is that a decryption committee avoids general-purpose MPC by using a specialized decryption protocol.

3.3 Design of Aero

We now go over the design details of Aero phase-by-phase. The main challenge in each phase is keeping the device overhead low while protecting against the malicious aggregator and the malicious subset of devices. We highlight these challenges, and Aero's key design choices and techniques.

But before proceeding, we briefly discuss committee formation, which is common to mul-

tiple phases. To form committees, Aero uses the sortition protocol from Orchard (which in turn used Algorand's protocol [73]). This protocol relies on a publicly verifiable source of randomness so that the results of the election are verifiable by all devices. At the end of the protocol, the aggregator publishes the list (public keys) of the committee members by putting it on the bulletin board. An important aspect of committee formation is committee size and the number of malicious devices in a committee:provision of a larger number of malicious devices A relative to the committee size C increases costs but ensures higher resiliency. Like Orchard, Aero makes a probabilistic argument [29] to select C and A such that the probability of the number of malicious devices exceeding A is small. For example, if the overall populationcontains up to $f = 3\%$ malicious devices (§3.1.1), then the probability that a randomly sampled subset of $C = 45$ devices contain more than $A = 2C/5 = 18$ malicious devices is less than $9.6 \cdot 10^{-14}$.

3.3.1 Setup phase

Much of Aero's setup phase is similar to Orchard. During this phase, (i) the aggregator samples the master committee, which then (ii) receives and validates inputs for the round (i.e., receives model parameters θ^t for the current round t, the device selection probability q, noise scale z, and clipping bound S, and generates new values of the DP parameters ϵ, δ), and (iii) generates keys for cryptographic primitives (§3.2.3). We do not focus on the first two pieces to avoid repetition with Orchard, but include them in the supplementary material for completeness (Appendix §A.1.5). Instead, the key challenge in Aero is the overhead of key generation.

Recall (§3.1.3) that Orchard uses MPC among the master committee members to correctly run the key generation function and ensure that even if the malicious members of the committee collude, they cannot recover the AHE secret key. The overhead of this MPC is high:

≈1 GiB of network transfers and 180 seconds of CPU time per committee device.How can this overhead be reduced?

One idea [30] is to reuse keys across rounds rather than generate them afresh for each round. Indeed, this is what Aero does: the master committee in round 1 generates the keys and shares them with the committee for the next round, and this committee then shares the keys with the committee for the third round, and so on. But one has to be careful.

Consider the following attack. Say that the malicious aggregator receives a victim device $k's$ update $Enc(pk, \Delta_k^t)$ in round t. Then, in the next round $t+1$, the aggregator colludes with a malicious device in the overall population to use $Enc(pk, \Delta_k^t)$ as the device's update. This attack enables the aggregator to violate differential privacy as the victim device's input does not satisfy the required clipping bound S in round $t+1$ due to its multiple copies (§3.2.1). Orchard does not suffer from this attack as it generates fresh keys in each round: the ciphertext for round t decrypts to a random message with round $t+1$'s key. However, prior work that reuses keys in this manner (in particular, Mycelium [30]) does suffer from this attack.

Thus, Aero must apply the reuse-of-keys idea with care. Aero adjusts the generate and add phases of its protocol (§3.2.3) to prevent the aforementioned attack. We are not in a position yet to describe these changes, but we will detail them shortly when we describe these other phases (§3.3.2, §3.3.3). Meanwhile, the changes in the setup phase relative to Orchard are the following: for the AHE secret key sk, Aero implements an efficient verifiable secret redistribution scheme [74, 30] such that committee members at round $t+1$ securely obtain the relevant shares of the key from the committee at round t. For the public keys (AHE public key pk, and both the ZK-proof public proving and verification keys), the committee for round t signs a certificate containing these keys and uploads it to the bulletin board, and the committee for round $t+1$ downloads it from the board.

The savings by switching from key generation to key resharing are substantial for the network, with a slight increase in CPU. While the MPC solution incurs ≈1 GiB of network

transfers and 180 seconds of CPU time per committee device, key resharing requires 125 MiB and 187 seconds, respectively (§3.5.2). The CPU is higher because key resharing requires certain expensive field exponentiation operations [74].

3.3.2 Generate phase

Recall from §3.2.3 that during this phase (i) Aero must pick a subset of devices to generate updates to the model parameters, (ii) the DP-noise committee must generate Gaussian noise for differential privacy, and (iii) both types of devices must encrypt their generated data (updates and noise).

Device sampling for updates. One design choice is to ask the aggregator to sample devices that will contribute updates. The problem with this option is that the (malicious) aggregator may choose the devices non-uniformly; for instance, it may pick an honest device more often than the device should be picked, violating differential privacy. An alternative is to ask the devices to sample themselves with probability q (as required by DP-FedAvg; line 8 in Figure 3.2). But then a malicious device may pick itself in every round, which would allow it to significantly affect model accuracy.

Aero adopts a hybrid and efficient design in which devices sample themselves but the aggregator verifies the sampling. Let B^t be a publicly verifiable source of randomness for round t; this is the same randomness that is used in the sortition protocol to sample committees for the round. Then, each device k with public key π_k computes $PRG(\pi_k||B^t)$, where PRG is a pseudorandom generator. Next, the device scales the PRG output to a value between 0 and 1, and checks if the result is less than q. For instance, if the PRG output is 8 bytes, then the device divides this number by $2^{64} - 1$. If selected, the device runs the UserUpdate procedure (line 10 in Figure 3.2) to generate updates for the round. This approach of sampling is efficient as devices only perform local computations.

Gaussian noise generation. The default option is to make the DP-noise committee generate the noise using MPC, but as noted several times in this paper, this option is expensive. Instead, Aero adapts prior work [75] on distributed Gaussian noise generation. The Gaussian distribution has the property that if an element sampled from $\mathcal{N}(0, a)$ is added to another element sampled from $\mathcal{N}(0, b)$, then the sum is a sample of $\mathcal{N}(a+b)$ [75, 52, 76]. This works well for the simple case when all C committee members of the DP-noise committee are honest. Given the standard deviation of the Gaussian distribution, $\sigma = z \cdot S$, the devices can independently compute their additive share. That is, to generate samples from $\mathcal{N}(0, I\sigma^2)$ (line 12 in Figure 3.2), each committee member can sample its share of the noise from the distribution $\mathcal{N}(0, I\frac{\sigma^2}{C})$.

The challenge in Aero is therefore: how do we account for the A malicious devices in the DP-committee? These devices may behave arbitrarily and may thus generate either no noise or large amounts of it. Adding unnecessary noise hurts accuracy, not privacy. In contrast, failing to add noise may violate privacy. We thus consider the worst case in which malicious users fail to add any noise and ask honest devices to compensate. Each honest client thus samples its noise share from the distribution $\mathcal{N}(0, I\frac{\sigma^2}{C-A})$.[2]

This algorithm generates noise cheaply without expensive MPC. The downside is that it may generate more noise than necessary, hurting accuracy. To mitigate this risk, we carefully choose the committee size to minimize the ratio of additional noise. Specifically, we choose C to keep the ratio $(C-A)/C$ close to 1. For instance, instead of picking a committee containing a few tens of devices similar to the master committee, we pick a somewhat larger DP-noise committee: $(A, C) = (40, 280)$.[3]

Encryption and ZK-proofs. Once the devices generate their updates or shares of the Gaus-

[2]Aero can further compensate for honest-but-offline devices. Say, for e.g., that B of $C - A$ honest devices must be provisioned to be offline. Then, Aero subtracts B from $C - A$ to get the number of honest-but-online devices.

[3]Using a probabilistic argument for committee size selection as before (§3.3), if $f = 3\%$ devices in the overall population are malicious, then the chances of sampling 280 devices with more than 1/7th malicious is $4.1 \cdot 10^{-14}$.

sian noise, they encrypt the content using the public key of the AHE scheme to prevent the aggregator from learning the content. Further, they certify using a ZK-proof scheme that the encryption is done correctly and the data being encrypted is bounded by the clipping value S (so that malicious devices may not supply arbitrary updates). This encryption and ZK-proof generation is same as in Orchard, but Aero requires additional changes. Recall from the setup phase that Aero must ensure a ciphertext generated in a round is used only in that round, to prevent complications due to reuse of keys (§3.3.1). To do this, each device concatenates the round number t (as a timestamp) to the plaintext message before encrypting it. Further, the ZK-proof includes additional constraints that prove that a prefix of the plaintext message equals the current round number.

3.3.3 Add phase

Recall that during the add phase (i) the aggregator adds ciphertexts containing device updates to those containing shares of Gaussian noise, (ii) the devices collectively verify the aggregator's addition (§3.2.3).

This work during the add phase has subtle requirements. So first, we expand on these requirements while considering a toy example with two honest and a malicious device. The first honest device's input is $\textsc{Enc}(pk, \Delta)$, where Δ is its update, while the second honest device's input is $\textsc{Enc}(pk, n)$, where n is the Gaussian noise. For this toy example, first (**R1**), the aggregator must not omit $\textsc{Enc}(pk, n)$ from the aggregate as the added noise would then be insufficient to protect Δ and guarantee DP. Second (**R2**), the aggregator must not let the malicious device use $\textsc{Enc}(pk, \Delta)$ as its input. Relatedly, the aggregator itself must not modify $\textsc{Enc}(pk, \Delta)$ to $\textsc{Enc}(pk, k \cdot \Delta)$, where k is a scalar, using the additively homomorphic properties of the encryption scheme. The reason is that these changes can violate the clipping requirement that a device's input is bounded by S (e.g., $2 \cdot \Delta$ may be larger than S). And,

third (**R3**), the aggregator must ensure that the above (the malicious device or the aggregator copying a device's input) does not happen across rounds, as recall that Aero uses the same encryption key in multiple rounds (§3.3.1).

One option to satisfy these requirements is to use the verifiable aggregation protocol of Orchard [29] that is based on summation trees. The main challenge is resource costs. Briefly, in this protocol, the aggregator arranges the ciphertexts to be aggregated as leaf nodes of a tree, and publishes the nodes of the tree leading to the root node. For example, the leaf nodes will be $\textsc{Enc}(pk, \Delta)$ and $\textsc{Enc}(pk, n)$, and the root node will be $\textsc{Enc}(pk, \Delta) + \textsc{Enc}(pk, n)$, for the toy example above. Then, devices in the entire population inspect parts of this tree: download a few children and their parents and check that the addition is done correctly, that the leaf nodes haven't been modified by the aggregator, and the leaf nodes that should be included are indeed included. The problem is that Orchard requires a device to download and check about $3 \cdot s$ nodes of the tree [29, 28], where s is a configurable parameter whose default value is six. But for realistic models, each node is made of many ciphertexts (e.g., the 1.2M parameter CNN model requires $\ell = 293$ ciphertexts), and 18 such nodes add to 738 MiB.

Aero improves this protocol using two ideas. First, Aero observes that the entire population of devices that must collectively check the tree is massive (e.g., 10^9). Besides, although the tree has bulky nodes with many ciphertexts, the total number of nodes is not high due to sampling (e.g., only 10,000 devices contribute updates in a round). Thus, Aero moves away from one summation tree with "bulky" nodes, to ℓ summation trees with "small" nodes, where ℓ is the number of ciphertexts comprising a device's update (e.g., $\ell = 293$ for the 1.2M parameter model). Then, each device probabilistically selects a handful of trees, and a checks few nodes within each selected tree.

Second, Aero optimizes how a device tests whether the sum of two ciphertexts equals a third ciphertext. Aero recognizes that ciphertexts can be expressed as polynomials and the validity of their addition can be checked efficiently using a technique called polynomial

identity testing (PIT) [77, 78]. Roughly, PIT says that the sum of polynomials can be checked by evaluating them at a random point and checking the sum of these evaluations. Using PIT, Aero replaces the ciphertexts at the non-leaf nodes of the summation trees with their much smaller evaluations at a random point.

We now describe Aero's protocol in detail, first without the PIT optimization, and then with it.

Incorporating finer-grained summation trees. Aero's protocol has three steps: commit, add, and verify (Figure 3.4). In the *commit* step, all devices commit to their ciphertexts before submitting them to the aggregator (line 1–3 in Figure 3.4). The aggregator publishes a Merkle tree of these commitments to the bulletin board. Committing before submitting ensures that a malicious device cannot copy and submit an honest device's input (requirement **R2** above). Similarly, this design ensures that the aggregator cannot change a device's input (again requirement **R2**).

In the *add* step, the aggregator adds the ciphertexts via summation trees. Specifically, if device updates have ℓ ciphertexts, the aggregator creates ℓ summation trees, one per ciphertext (line 6 in Figure 3.4). The leaf vertices of the j-th tree are the j-th ciphertexts in the devices' inputs, while each parent is the sum of its children ciphertexts, and the root is the j-th ciphertext in the aggregation result. The aggregator publishes the vertices of the summation trees on the bulletin board (line 7 in Figure 3.4), allowing an honest device to check that its input is not omitted (requirement **R1** above).

In the *verify* step, each device in the system selects $q \cdot \ell$ summation trees, where q is the device sampling probability (line 8 in Figure 3.4), and checks s leaf nodes and $2s$ non-leaf nodes in each tree. ($s = 6$ in our implementation.) Specifically, the device checks that the leaf node ciphertexts are committed to in the commit step (requirement **R2**), and the ZK-proofs of the ciphertexts are valid, e.g., the first part of the plaintext message in the ciphertexts equals

the current round number (requirement **R3**). For the non-leafs, the device checks that they sum to their children.

Incorporating PIT. Checking the non-leaf vertices is a main source of overhead for the protocol above. The reason is that even though each non-leaf is a single ciphertext, this ciphertext is large: for the quantum-secure AHE scheme Aero uses (§3.4), a ciphertext is 131 KiB, made of two polynomials of 2^{12} coefficients each, where each coefficient is 16 bytes.

As mentioned earlier, Aero reduces this overhead by using polynomial identity testing (PIT) [77, 78]. This test says that given a d-degree polynomial $g(x)$ whose coefficients are in a field $\mathbb{F}$, one can test whether $g(x)$ is a zero polynomial by picking a number $r \in \mathbb{F}$ uniformly and testing whether $g(r) == 0$. This works because a d-degree polynomial has at most d solutions to $g(x) == 0$ and d is much less than $|\mathbb{F}|$.

Using PIT, Aero replaces the ciphertexts at the non-leafs with their evaluations at a random point r. Then, during the "Verify" step, a device checks (line 10 in Fig. 3.4) whether these evaluations (rather than ciphertexts) add up. Thus, instead of downloading three ciphertexts with $2 \cdot 2^{12}$ field elements each, a device downloads 2 elements of $\mathbb{F}$ per ciphertext.

A requirement for PIT is generation of r, which must be sampled uniformly from the coefficient field. For this task, Aero extends the master committee to publish an r to the bulletin board in the add step, using a known protocol to securely and efficiently generate a random number [79, 66].

Add phase protocol of Aero

Commit step

1. Each device (with public key π_i) that generates ciphertexts $(c_{i1}, \ldots, c_{i\ell})$ and corresponding ZK-proofs $(z_{i1}, \ldots, z_{i\ell})$ in the generate phase does the following: for each $j \in [1, \ell]$, generates a commitment to (π_i, c_{ij}), namely $t_{ij} = Hash(r_{ij}||c_{ij}||\pi_i)$, where r_{ij} is a random 128-bit nonce. The device sends $(\pi_i, t_{i1}, ..., t_{i\ell})$ to the aggregator $\mathcal{A}$.
2. $\mathcal{A}$ checks that $PRG(\pi_i||B^t) \leq q$ or π_i is a member of the DP-noise committee. Otherwise, it ignores the message.
3. For each $j \in [1, \ell]$, $\mathcal{A}$ sorts pairs (π_i, t_{ij}) by π_i to form an array of tuples $Commit_j$. $\mathcal{A}$ then generates a Merkle tree MC_j from array $Commit_j$ and publishes the root to the bulletin board.
4. Each device from step 1 above sends $(\pi_i, c_{ij}, r_{ij}, z_{ij})$ to the aggregator, for each $j \in [1, \ell]$.
5. $\mathcal{A}$ checks the messages. If any proof z_{ij} or any commitment $t_{ij} = Hash(r_{ij}||c_{ij}||\pi_i)$ is incorrect, it ignores the message.

Add step

6. For each $j \in [1, \ell]$, $\mathcal{A}$ computes a summation tree ST_j. The leaves are $ST_j(0, i) = (\pi_i, c_{ij}, r_{ij}, z_{ij})$ if $\mathcal{A}$ got a correct message and $(\pi_i, \perp)$ otherwise. Each non-leaf vertex has two children and a parent ciphertext is the sum of its children ciphertexts.
7. For each $j \in [1, \ell]$, $\mathcal{A}$ serializes all vertices of ST_j into an array and publishes a Merkle tree MS_j, as well as the root of the summation tree ST_j. To each device that sent a correct leaf, $\mathcal{A}$ also sends a proof that this leaf is in MS_j.

Verify step

Every device in the system does the following:

8. With probability q, select elements from $[1, \ell]$ to form an index array idx containing indices of the trees to inspect.
9. Choose a random $v_{init} \in [0, M' - 1]$. For each $j \in idx$, and for each $i \in [v_{init}, v_{init} + s) \mod M'$,
   (a) verify that $ST_j(0, i) = (\pi_i, \perp)$ or $(\pi_i, c_{ij}, r_{ij}, z_{ij})$, and $\pi_i \leq \pi_{i+1}$ (except if $i = M' - 1$)
   (b) if $ST_j(0, i) \neq (\pi_i, \perp)$, verify that commitment t_{ij} appears in MC_j, $t_{ij} = H(r_{ij}||c_{ij}||\pi_i)$, and z_{ij} is valid
   (c) check $ST_j(0, i)$ is in MS_j
10. For each $j \in idx$, choose s distinct non-leaf vertices of ST_j. For each such non-leaf vertex, verify that its ciphertext equals the sum of the children ciphertexts, and that the vertex and its children are in MS_j.

Figure 3.4: Aero's verifiable aggregation. This description does not include the PIT optimization (described in text) that applies to line 10.

3.3.4 Release phase

During the release phase, Aero must decrypt the ℓ ciphertexts from the add phase, i.e., the ℓ root nodes of the ℓ summation trees. The default, but expensive, option is to use MPC among the members of the decryption committees.

Aero addresses this efficiency challenge using known ideas and applying them; i.e., Aero's contribution in this phase is not new techniques, but the observation that existing ideas can be applied. Nevertheless, applying these ideas requires some care and work.

First, recall that Aero has multiple decryption committees (§3.2.2). Naturally, to reduce per-device work, each committee decrypts a few of the ℓ ciphertexts. A design question for Aero is how many committees should it use. On the one hand, more committees are desirable (best case is ℓ). However, more committees also mean that each has to be larger to ensure that none of them samples more than A out of C malicious devices, breaking the threshold assumptions of a committee. Meanwhile, a larger committee means more overhead. In practice (§3.5), we take a middle ground and configure Aero to use ten decryption committees.

Second, Aero reduces each committee's work relative to the MPC baseline, using a fast distributed decryption protocol to decrypt the ciphertexts [80]. The use of this protocol is possible as a decryption committee's task is only of decryption given how we formed and assigned work to different types of committees (§3.2.2). This fast protocol requires the committee devices to mainly perform local computations with little interaction with each other. The caveat is that for this protocol to be applicable, the committee members must know an upper bound on the number of additive homomorphic operations on the ciphertexts they are decrypting.[4] Fortunately, in Aero's setting this bound is known: it is the maximum number of devices whose data the aggregator adds in the add phase (M_{max} in Figure 3.4). The benefit of distributed decryption (and moving work outside MPC) meanwhile is substantial.

[4]This bound is needed to add a "smudging noise" to the committee's decryption output to ensure that the output does not leak information on the inputs to the aggregation [81].

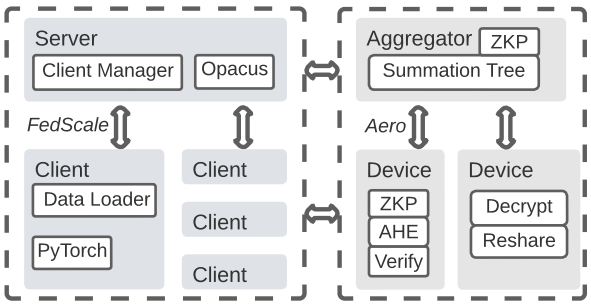


Figure 3.5: An overview of Aero's implementation.

3.3.5 Privacy proof

Aero's protocol provides the required differential privacy guarantee (§3.1.2). The supplementary material (Appendix A.1) contains a proof. But, briefly, the key reasons are that (i) in the generate phase, honest devices sample themselves to make sure that they are not sampled more than expected, (ii) the verifiable aggregation protects these devices' input, and (iii) key resharing and fast decryption protocols keep secret keys hidden.

3.4 Prototype implementation

We implemented a prototype of Aero atop FedScale [50], which is a scalable system for federated learning capable of handling a large number of devices. By default, FedScale supports algorithms such as FedAvg and FedSGD (without differential privacy). Further, it allows a data analyst to specify the model using the popular PyTorch framework.

Our Aero prototype extends FedScale in the following way (Figure 3.5). First, it extends the programming layer of FedScale with Opacus [82], which is a library that adjusts a PyTorch model to make it suitable for differentially private federated learning; for instance, Opacus replaces the batch normalization layer of a neural network with group normalization. Second, our prototype extends the device-side code of FedScale with additional components needed for the various committees and phases in Aero (key resharing, Gaussian noise generation,

Dataset	Model	Size	FedScale	Aero
FEMNIST [59]	LeNet [83]	49K	75%	74%
	CNND [56]	1.2M	78%	68%
	CNNF [48]	1.7M	79%	68%
	AlexNet [84]	3.9M	78%	40%
CIFAR10 [85]	LeNet [83]	62K	48%	48%
	ResNet20 [86]	272K	59%	48%
	ResNet56 [86]	855K	54%	35%
Speech [87]	MobileNetV2 [88]	2M	57%	4%

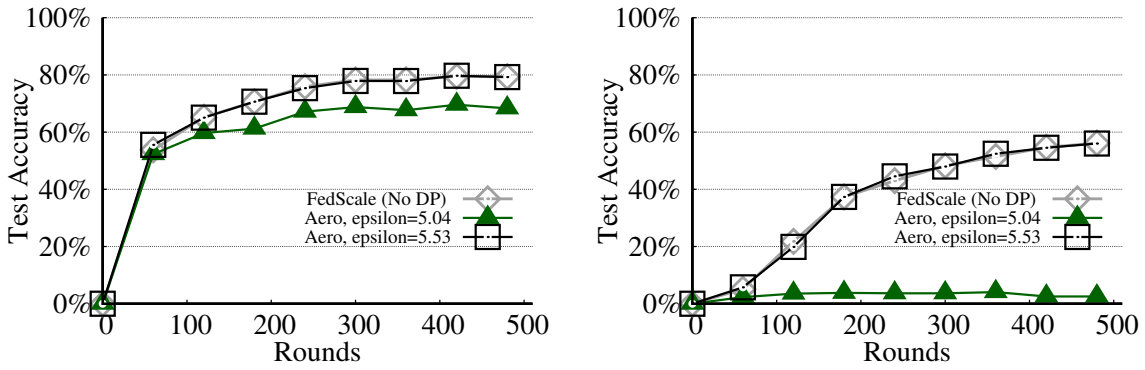
Figure 3.6: Test accuracy for different models after 480 rounds of training and differential privacy parameters (ϵ, δ) set to (5.03, $W^{-1.1}$). As shown later, increasing ϵ can recover the accuracy loss.

verifiable aggregation, and distributed decryption; §3.3). FedScale is written in Python while the code we added is in Rust; thus, we use PyO3 to wrap the Rust code with Python interfaces. Third, our prototype extends the FedScale server-side code with Aero's aggregator code and the code to coordinate the various phases of Aero. In total, we added ≈4,300 lines of Rust to FedScale.

Our prototype configures the cryptographic primitives for 128-bit security. For additively homomorphic encryption, we use the BFV encryption scheme. We set the polynomial degree in BFV to 2^{12} and use the default parameters from Microsoft SEAL [89]. For ZK-proofs, we use ark_groth16 [90], which implements the zkSNARK of Jens Groth [47].

3.5 Evaluation

We evaluate Aero in two parts. First, we compare it with plain federated learning, specifically, the FedScale system. This comparison sheds light on the cost of privacy both in terms of model accuracy and resource consumption on the devices. Second, we compare Aero to Orchard, which is the state-of-the-art system for training models in a federated manner in the same threat model as Aero. This comparison helps understand the effectiveness of Aero's techniques in reducing overhead. Our main results are the following:



(a) CNNF (b) MobileNetV2

Figure 3.7: Test accuracy versus rounds for Aero and FedScale for the 1.7M parameter CNNF model and the 2M parameter MobileNetV2 model.

- Aero can train models with comparable accuracy to FedScale (plain federated learning). For instance, for a CNN model over the FEMNIST dataset, Aero produces a model with 79.2% accuracy with DP parameter $\epsilon = 5.53$, relative to 79.3% in FedScale, after 480 rounds of training.
- Aero's CPU and network overhead is low to moderate: for a 1.2M parameter model, devices spend 15 ms of CPU and 3.12 KiBs of network transfers most of the time, and occasionally (with a probability of 10^{-5} in a round) 13.4 min. of processor time and 234 MiBs of network transfers.
- Aero's techniques improve over Orchard by up to $2.3 \cdot 10^5\times$.

Testbed. Our testbed has machines of type `c5.24xlarge` on Amazon EC2. Each machine has 96vCPUs, 192 GiB RAM, and 25 Gbps network bandwidth. We use a single machine for running Aero's server. Meanwhile, we co-locate multiple devices on a machine: each device is assigned six CPUs given that modern mobiles have processors with four to eight CPUs.

Default system configuration. Unless specified otherwise, we configure the systems to assume $W = 10^9$ total devices. For Aero, we set the default device sampling probability q in DP-FedAvg to 10^{-5}; i.e., the expected number of devices that contribute updates in a round

is 10^4. We also configure Aero to use ten decryption committees, where each committee has a total of $C = 45$ devices of which $A = 18$ may be malicious. The first decryption committee also serves as the master committee. We configure the DP-noise committee with $(A, C) = (40, 280)$. For Orchard, we configure its committee to have 40 devices of which 16 may be malicious.[5]

3.5.1 Comparison with FedScale

Accuracy. We evaluate several datasets and models to compare Aero with FedScale, specifically, the FedAvg algorithm in FedScale. Figure 3.6 shows these datasets and models. We use CNND and CNNF for two different CNN models: one dropout model [56] and the other from the FedAvg paper [48].

Aero's accuracy depends on the DP parameters ϵ and δ. For Figure 3.6 experiments, we set $\epsilon = 5.04$ and $\delta = 1/W^{1.1}$. Further, for both systems, we set all other training parameters (batch size, the number of device-side training epochs, etc.) per the examples provided by FedScale for each dataset.

Figure 3.6 compares the accuracies after 480 rounds of training (these models converge in roughly 400-500 rounds). Generally, Aero's accuracy loss grows with the number of model parameters. The reason is that DP-FedAvg adds noise for every parameter and thus the norm of the noise increases with the number of parameters.

Although Aero's accuracy loss is (very) high for a larger number of parameters, this loss is recoverable by increasing ϵ (but still keeping it at a recommended value). Figure 3.7 shows accuracy for two values of ϵ for two example models. Increasing ϵ from 5.04 to 5.53 recovers the accuracy loss. For instance, for the CNNF model, FedScale's accuracy is 79.3% after 480 rounds, while Aero's is 79.2%. The reason is that as ϵ increases, more devices can con-

[5] Aero's committees are larger because it must ensure, using a union bound, that the chance of sampling more than A malicious devices across *any* of its committees is the same as in Orchard.

		CPU (ms)		network (KiB)	
Model	Size	FedScale	Aero	FedScale	Aero
LeNet	49K	3.36E-4	2	3.93E-5	0.96
CNND	1.2M	9.50E-4	55	9.66E-4	3.87
CNNF	1.7M	9.49E-4	77	1.35E-3	5.15
AlexNet	3.9M	1.75E-3	170	3.12E-3	11.0

Figure 3.8: Per device per round average cost for different models.

tribute updates (q increases), which increases the signal relative to the differential privacy noise. Overall, Aero can give competitive accuracy as plain federated learning for models with parameters ranging from tens of thousand to a few million.

Device overhead. Another cost of privacy relative to plain federated learning is increased device overhead. Figure 3.8 summarizes the average CPU and network cost per round per device for the four models on the FEMNIST dataset. (We picked the FEMNIST dataset just as an example, but the results for the other datasets are qualitatively the same.)

Overall, an Aero device on average (considering the different types of Aero devices) spends $5.9 \cdot 10^3 - 9.7 \cdot 10^4\times$ higher CPU and $3.5 \cdot 10^3 - 2.4 \cdot 10^4\times$ higher network relative to FedScale. This overhead is due to the fact that FedScale does not use any cryptographic operations, while Aero devices use many, for example, encryption and ZK-proofs during the generate phase, and verifiable aggregation during the add phase. However, Aero's overhead, at least, on average, is low (Figure 3.8). Further, as we will show next, Aero's worst-case overhead is also moderate.

3.5.2 Comparison to Orchard

Both Aero and Orchard have multiple types of devices. Aero has devices that participate in the master committee, generate updates (or Gaussian noise), verify the aggregator's work, and participate in the decryption committee. Similarly, Orchard has generator, verifier, and committee devices. We compare overhead for these devices separately.

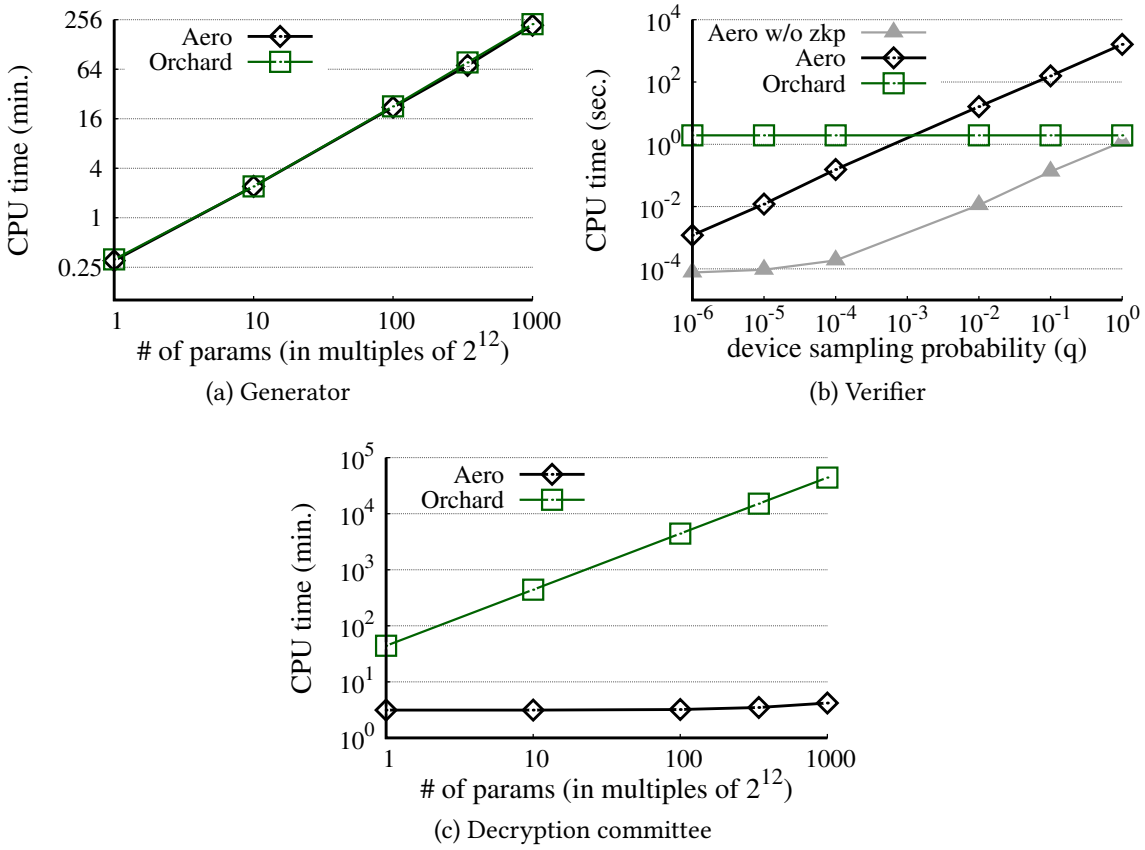


Figure 3.9: CPU time per device per round of training for different device roles in Aero and Orchard.

Generator device overhead. The overhead for the generators changes only with the model size (after excluding the training time to generate the plain updates). Thus, we vary the number of model parameters and report overhead.

Figure 3.9a shows the CPU time and Figure 3.10a shows the network transfers with a varying number of model parameters. These overheads grow linearly with the number of model parameters (the network overhead is not a straight line as it includes a fixed cost of 60 MiB to download ZK-proof proving keys).The reason is that the dominant operations for a generator device are generating ZK-proofs and shipping ciphertexts to the aggregator. The number of both operations is proportional to the number of parameters (§3.3.2).

In terms of absolute overhead, a specific data point of interest is a million-parameter

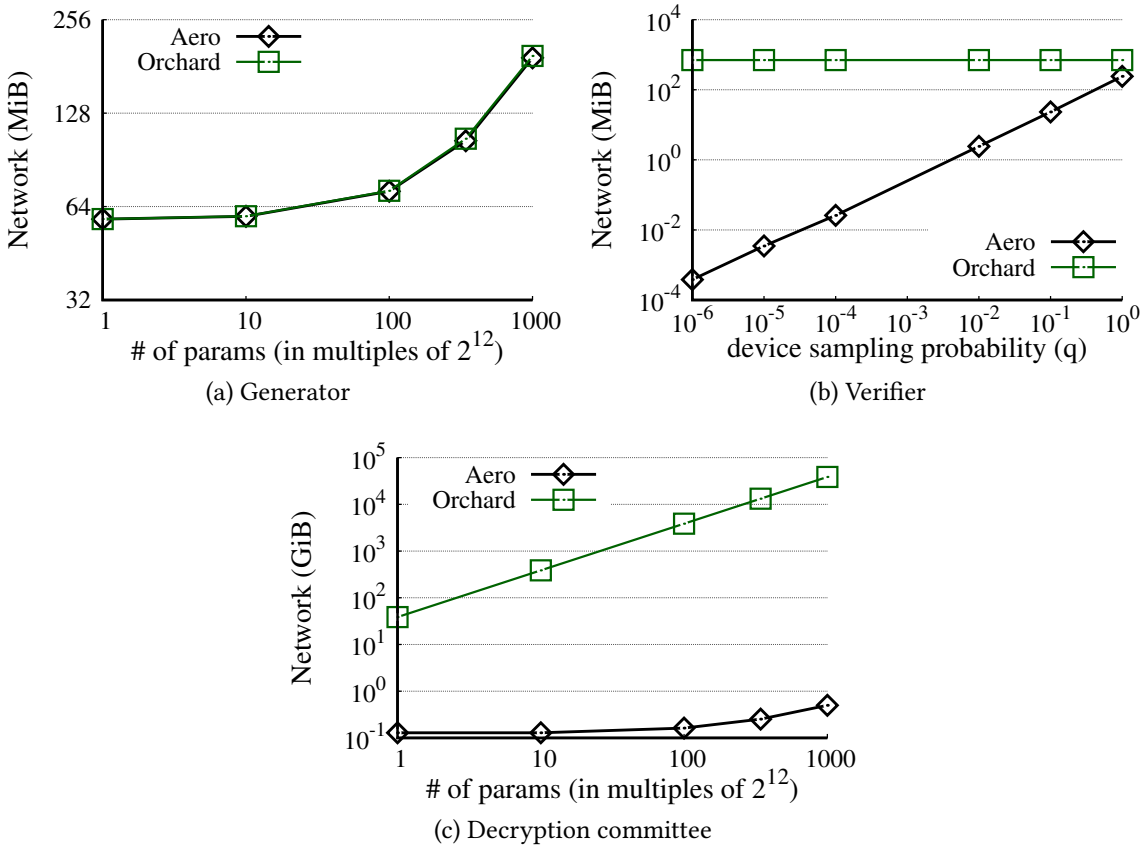


Figure 3.10: Network transfers per device per round of training for different device roles in Aero and Orchard.

model, e.g., the CNND model with 1.2M parameters. For this size, a generator device spends 1.01 hours in CPU time, or equivalently 13.4 minutes of latency (wall-clock time) over six cores. The generator also sends 101 MiB of data over the network. These overheads are moderate, considering the fact that the probability that a device will be a generator in a round is small: 10^{-5}.

Finally, the CPU and network overhead for Aero and Orchard is roughly the same.The reason is that the dominant operations for the two systems are common: ZK-proofs and upload of ciphertexts.

Verifier device overhead. Figure 3.9b shows CPU and Figure 3.10b shows network overhead for the verifier devices that participate in the verifiable aggregation protocol (§3.3.3). These

experiments fix the number of model parameters to 1.2M and vary the probability q with which a verifier device samples summation trees to inspect (recall that a verifier device in Aero checks $q \cdot \ell$ summation trees). For Orchard, overhead does not change with q (q is effectively 1).

Overall, Aero's verifier devices, which are the bulk of the devices in the system, are efficient consuming a few milliseconds of CPU and a few KiBs of network transfers. For instance, for $q = 10^{-5}$, Aero incurs 3.12 KiB in network and 15 ms of CPU time, while Orchard incurs 1.96 seconds (130×) and 738 MiB ($2.36 \cdot 10^5\times$).

Comparing Aero with Orchard, a verifier in Aero consumes lower CPU than Orchard for smaller values of q but a higher CPU for larger q. This trend is due to constants: even though an Aero device checks $q \cdot \ell$ summation trees and $3s$ ciphertexts in each tree versus $\ell \cdot 3s$ ciphertexts in Orchard, Aero devices verify the ZK-proofs to address the reuse-of-keys issue, while Orchard does not have such a requirement (§3.3.3). Each proof check takes ≈700 ms on a single CPU of `c5.24xlarge`. Indeed, Aero w/o ZK-proof check (another line in the plot) is strictly better than Orchard.

Aero's network overhead increases linearly with q, while Orchard's stays constant as it does not do sampling (Figure 3.10b). Notably, when $q = 1$, i.e., when Aero and Orchard check the same number of ciphertexts, a Aero verifier consumes 251 MB, which is $\approx 1/3$rd of Orchard. This is because polynomial identity testing allows a Aero verifier to download evaluations of ciphertext polynomials rather than the full polynomials from non-leaf vertices (§3.3.3).

Committee device overhead. Figure 3.9c and Figure 3.10c show the CPU and network overhead of decryption committee devices as a function of the model size. (In Aero, the first decryption committee also serves as the master committee.)

Aero's overheads are much lower than Orchard's—for 1.2M parameters, CPU time is 206 s

in Aero versus 214 hours in Orchard (i.e., $3751\times$ lower), and network is 234 MiB in Aero versus 11 TiB in Orchard (i.e., $4.8 \cdot 10^4\times$ lower). This improvement is for two reasons. First, Aero divides the decryption of multiple ciphertexts across committees, and thus each performs less work. Second, Aero uses the distributed decryption protocol (§3.3.4), while Orchard uses the general-purpose SCALE-MAMBA MPC [65].

3.6 Related work

Aero's goal is to add the rigorous guarantee of differential privacy to federated learning—at low device overhead. This section compares Aero to prior work with similar goals.

Local differential privacy (LDP). In LDP, devices *locally* add noise to their updates before submitting them for aggregation [11, 12, 13, 14, 15, 16, 91, 92, 93, 94, 57, 95, 96, 97, 98, 99, 100, 101, 102]. On the plus side, the privacy guarantee in LDP does not depend on the behavior of the aggregator, as devices add noises locally. Further, LDP is scalable as it adds small device-side overhead relative to plain federated learning. However, on the negative side, since each device perturbs its update, the trained model can have a large error.

Central differential privacy (CDP). Given the accuracy loss in LDP, many systems target CDP [103, 104, 105, 75, 52, 106, 107, 108, 19, 29, 28, 109]. The core challenge is of hiding sensitive device updates from the aggregator.

Several systems in this category target a setting of a few tens of devices to a few thousand devices [109, 52, 75, 108, 105]. These systems require *all* devices to participate in one or more cryptographic primitives, and thus their overhead grows with the number of devices. For example, in secure aggregation based FLDP [108], each device generates a secret key, then masks its update using the key, before sending the masked update to the aggregator. Then, the devices securely sum their masks to subtract them from the aggregator's result. This latter protocol requires each device to secretly share its mask with all others.

Chase et al. [106] do not require their protocol to scale with the number of devices: two devices aggregate updates from all others before generating and adding DP noise via Yao's MPC protocol [63]. The issue is that if the adversary compromises the two devices, it learns the updates.

Honeycrisp [29], Orchard [28], and Mycelium [30] target a setting of a billion devices. One of their key insights is to run expensive cryptographic protocols among a small, randomly-sampled committee, while leveraging an untrusted resourceful aggregator to help with the aggregation. Among the three systems, Orchard supports learning tasks, while Honeycrisp supports aggregate statistics and Mycelium supports graph analytics. The limitation of Orchard is that it imposes a large overhead on the devices (§3.1.3, §3.5). Aero improves over Orchard by several orders of magnitude (§3.5).

An alternative to cryptography is to use trusted hardware, e.g., Intel SGX [19]. These systems add negligible overhead over plain federated learning, but trusting the hardware design and manufacturer is a strong assumption [20, 21, 22].

No differential privacy. Many systems provide a weaker notion of privacy than differential privacy, for functionality such as federated machine learning [23, 24, 25, 26, 27, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133], statistics [134], and aggregation [135, 136, 137, 138, 139]. For instance, BatchCrypt [114] uses Paillier AHE [140] to hide updates from the aggregator. The promise is that the adversary learns only the aggregate of the data of many devices. The fundamental issue is that aggregation does not provide a rigorous guarantee: one can learn individual training data from the trained model parameters [8, 9, 141, 10].

3.7 Summary

Federated learning over a large number of mobile devices is getting significant attention both in industry and academia. One big challenge of current practical systems, those that provide good accuracy and efficiency, is the trust they require: the data analyst must say "let's trust that the server will not be compromised". Aero adds an alternative. It shows that one can perform FL with good accuracy, moderate overhead, and the rigorous guarantee of differential privacy without trusting a central server or the data analyst. Aero improves the trade-off by focusing on a specific type of learning algorithms and tuning system architecture and design to these algorithms (§3.3). The main evaluation highlight is that Aero has comparable accuracy to plain federated learning, and improves over prior work Orchard that has strong guarantees by five orders of magnitude (§3.5).

Chapter 4

Colo

In this chapter, we describe an efficient system for our second type of target queries, privacy-preserving federated graph analytics in a strong threat model.

As a motivating example, consider the following scenario between a mobile app maker of a contact tracing application for an infectious disease like COVID-19 [142, 143, 144], and an influential data analyst such as the Centers for Disease Control and Prevention (CDC) [145]. The app maker installs the app on a large number of mobile devices, where it collects information on whether a device owner is currently infected, and when, where, and for how long the device comes in contact with other devices. Abstractly, one can view the devices as a graph where they are the nodes and their interactions are the edges. Meanwhile, the analyst wants to use the data to study disease patterns. For instance, it wants to understand the prevalence of superspreaders by evaluating the average number of infected devices in an infected device's neighborhood [34, 35]. *Can we enable the analyst to run such queries and learn their result? Further, can we do it in a way that doesn't require moving device and edge data outside the devices to a centralized location? And further still, can we ensure that only the query result is revealed and no new individual device information is learned by any other party?*

This is the problem of *privacy-preserving federated graph analytics*. The federated aspect of

this problem emphasizes keeping raw data at the devices, in contrast to centralizing the data, which is highly susceptible to data breaches, especially in bulk [3, 4, 5, 6]. Meanwhile, the privacy guarantee of the problem emphasizes that an entity should get only the information that it absolutely needs. Thus, an analyst may learn the query result that is an aggregate across devices. And any device may not learn any more information than it knows locally through its own data and edges.

In this scenario, we consider a threat model where devices and the analyst can be malicious as the data is highly sensitive and compromising devices or the analyst can often happen. Similar to prior work [30], we primarily focus on privacy: we guarantee privacy in the presence of malicious adversaries, but ensure integrity only when all parties are semi-honest. Ensuring malicious integrity would impose much overhead and make the system impractical. Thus we leave ensuring malicious integrity as future work.

As we discuss in related work (§4.6), privacy-preserving federated graph analytics is an emerging area of research and displays a trade-off between generality, privacy, and efficiency.

For instance, Gunther et al. [146, 147] have built a system RIPPLE to answer epidemiological questions. However, their system answers aggregation queries only where a device can securely sum its state (e.g., an integer) with its neighbors' state. It doesn't support other secure operations such as multiplication and comparsion. Thus, it cannot answer our motivating query on superspreaders. RIPPLE also releases partial sums to the devices, and does not consider the strongest of threat models as a set of its aggregators are honest-but-curious (but not malicious).

In contrast, Roth et al. [30] have built a general-purpose system, Mycelium, that assumes a strong threat model where both the devices and a centralized aggregator can be malicious.

However, Mycelium is expensive. For the superspreader query over 1M devices, each device incurs 8.79 hours of local CPU time and 5.73 GiB of network transfers.

Colo allows an analyst to run simple queries (like the superspreader query) that have

predicates with a limited set of inputs and outputs, and that evaluate these predicates between a device and its neighbors and then aggregate the results across the devices. Colo guarantees privacy (only the analyst learns the query result and no entity gets any other intermediate data) while assuming malicious devices and a set of M aggregation servers of which a fraction f of them can be malicious. A possible configuration from existing work [148, 149] is to set M to be 40-100 and $f = 0.2$. Finally, Colo is scalable and efficient: it supports a large number of devices in the order of a few million, while requiring them to contribute a small amount of CPU and network.

At a high level, Colo follows a workflow of local evaluation followed by a global aggregation across devices (§4.2). In the local evaluation, each device evaluates the query between itself and its neighbors. The global aggregation then aggregates these per-device outputs. In this workflow, Colo must address two challenges. First, it must hide node, edge, and topology data during the local evaluation without imposing a large overhead on the devices. Second, it must aggregate per-node outputs across the population of devices without revealing the intermediate results, and again while being efficient for the devices.

The first challenge of hiding node, edge, and topology data is tricky, especially with malicious devices (§4.1.4). For instance, say two neighboring devices v_A and v_B want to compute the product $v_A.inf \cdot v_B.inf$ (as needed for the superspreader query), where inf indicates their infection status. Then, a malicious device, say v_B, may set its infection status to $v_B.inf = 10^8$. As a result, the query result secretly encodes v_A's infection status (result is large if $v_A.inf = 1$). One may use a general-purpose tool from cryptography to address this issue, but that would be expensive. Furthermore, even if there were an efficient protocol for this computation, say that requires a single interaction between v_A and v_B, then this protocol must also hide that v_A and v_B are communicating, to protect their topology data, i.e., the fact they are neighbors.

Colo addresses the first challenge, particularly, the part of hiding node and edge data,

through a new, tailored secure computation protocol (§4.3.3). Colo observes that the query predicates that Colo targets operate over a limited set of inputs and produce a limited set of outputs. For instance, the legitimate inputs and outputs for $v_A.inf \cdot v_B.inf$ are all either zero or one. Thus, instead of using a general purpose secure computation protocol such as Yao's Garbled Circuits [150] that operates over arbitrary inputs and outputs, Colo uses a protocol that operates over a limited set of inputs and outputs. Specifically, one party, say v_B, computes all possible legitimate query outputs in plaintext, and then allows the other party v_A to pick one of these outputs privately using oblivious transfers (OT) [151, 44]. Colo fortifies this protocol against malicious behavior of v_B by incorporating random masks, efficient commitments [152, 153], and range proofs [70, 154] (§4.3.3).

The protocol described above doesn't yet address the requirement of hiding the topology of devices. To hide this data efficiently, that is, the knowledge of who is a neighbor with whom, Colo prohibits the devices from directly interacting with each other. Rather, Colo arranges for them to communicate via a set of servers, specifically, a set of 40 to 100 servers, where up to 20% are malicious (§4.2). This arrangement enables the servers to run a particular metadata hiding communication system, Karaoke [148], that is provably secure and low cost for the devices (although with significant overhead for the resourceful servers) (§4.3.3).

Finally, Colo addresses the second challenge of global aggregation across devices through straightforward secret sharing techniques while piggybacking on the set of servers (§4.2, §4.3.4). Specifically, devices add zero sum masks to their local outputs, and send shares of these local results to the servers. As long as one of the servers is honest, the analyst learns only the query output.

We have implemented (§4.4) and evaluated (§4.5) a prototype of Colo. Our evaluation shows that for 1M devices connected to at most 50 neighbors each, and for a set of example queries (Figure 4.1) which includes the superspreader query, a device in Colo incurs less than 8.4 minutes of (single core) CPU time and 4.93 MiB of network transfers. In contrast, the

Mycelium system of Roth et al. [30] requires a device-side cost of 8.79 hours (single core) CPU time and 5.73 GiB network transfers. In addition, Colo's server-side cost, depending on the query, ranges from $158 to $1,504 total for Colo's 40 servers (the lower number is for the superspreader query). In contrast, Mycelium's server-side cost is over $\$57,490$ per query.

Colo's limitations are substantial. In particular, it does not handle general purpose queries, rather only those that evaluate predicates over a bounded set of inputs and outputs. However, Colo scales to a significant number of devices and is efficient for them, in a strong threat model. But more importantly, unlike prior work, Colo shows that privacy-preserving federated graph analytics can be practical, and that the CDC *could* run certain queries over the devices' data while guaranteeing privacy in a strong sense, without draining the devices' resources, and without aggressively depleting its own budget (e.g., running the superspreader query in a large city every two weeks would cost around four thousand dollars annually).

4.1 Problem statement

4.1.1 Scenario

We consider a scenario consisting of a data analyst $\mathcal{A}$ and a large number of mobile devices v_i for $i \in [0, N)$. For instance, $N = 10^6$.

The devices form a graph. As an example, they may run a contact tracing application for COVID-19 [142, 143, 144] that collects information on the infection status of the device owners and identities of the devices they come in contact with, that is, their neighbors. More precisely, a device may have (i) *node data*, for example, a status variable inf indicating whether the device's owner is currently infected, $tInf$ indicating the time the owner got infected (or null if the owner is not infected), and demographic information such as age and ethnicity; (ii) *edge data*, for example, the number of times the device came in contact with a neighbor ($contacts$),

the cumulative duration ($duration$) of these interactions, and ($location$, $time$, $duration$) of each interaction; and, (iii) *topology data*, for example, the list of the device's neighbors.

Query	Description
Q1	The total number of infections in an infected participant's neighborhood *SELECT COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf*
Q2	The amount of time neighbor has spent near infected device if neighbor is infected within 5-15 days of contact with the device *SELECT SUM(edge.duration) FROM neigh(1) WHERE self.inf & neighbor.inf & (neighbor.tInf ∈ [edge.lastContact+5 days, edge.lastContact+15 days])*
Q3	The frequency of contact between device and neighbor, if device infected neighbor *SELECT SUM(edge.contacts)/COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf & (neighbor.tInf > self.tInf+2 days)*
Q4	Secondary attack rate of infected devices if they traveled on the subway *SELECT SUM(neighbor.inf)/COUNT(*) FROM neigh(1) WHERE self.inf & onSubway(edge.lastContact.location)*
Q5	The number of secondary infections caused by infected devices in different age groups *SELECT COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf & (neighbor.tInf > self.tInf+2 days) GROUP BY self.age*
Q6	The number of secondary infections based on type of exposure (such as family, social, work) *SELECT COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf & (neighbor.tInf > self.tInf+2 days) GROUP BY edge.setting*
Q7	Secondary attack rates in household vs non-household contacts *SELECT SUM(neighbor.inf)/COUNT(*) FROM neigh(1) WHERE self.inf GROUP BY isHousehold(edge.lastContact.location)*
Q8	Secondary attack rates within case-contact pairs in the same age group *SELECT SUM(neighbor.inf)/COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.age∈[0,100] & self.age∈[neighbor.age-10,neighbor.age+10]*

Figure 4.1: Example graph queries from Mycelium [30] and the literature on health analytics [34, 155, 156, 35, 157, 158, 159, 160, 161]. We assume that the domain of the inputs to these queries is bounded, for example, $inf \in [0, 1]$ and $tinf \in [1, 120]$, referring to the days in the latest few months.

The analyst $\mathcal{A}$, say a large hospital or CDC in the context of the contact tracing application, wants to analyze the device data by running graph queries. Figure 4.1 shows a few example queries from the literature [34, 155, 156, 35, 157, 158, 159, 160, 161] (these queries are a subset of the ones considered in the Mycelium system of Roth et al. [30]). For instance, $\mathcal{A}$ may want to learn the number of active infections infected devices have in their neighborhood (Q1). As

another example, it may want to learn how frequently infected devices contact their neighbors who are subsequently infected (Q3). Generally, these queries perform a local computation at every device and its neighborhood, and then aggregate the per-device results.

4.1.2 Threat model

We assume that the devices are malicious, i.e., an adversary can compromise an arbitrary subset of devices. A compromised device may try to learn information about an honest device beyond what it knows from its own data. For instance, if a compromised device is a neighbor of an honest device, then the compromised device already has edge data for their edge (e.g., the location and time of their last meeting); however, the adversary may further want to learn if the honest device is infected (neighbor's node data), whether it recently met someone on the subway (neighbor's edge data), and whom it recently came in contact with (neighbor's topology data).

The analyst may also be malicious and want to learn about individual device node, edge, or topology data—information that is more granular than the result of the queries.

Finally, we assume that the adversary may also observe and manipulate network traffic, for instance, in the backbone network, and try to infer relationships between devices.

4.1.3 Goals and non-goals

Target queries. Ideally, we would support arbitrary graph queries. However, as noted earlier, generality of queries is in tension with privacy and efficiency. Thus, in this paper we focus on simpler queries such as those in Figure 4.1 where the aggregations across devices are SUM, COUNT and AVG operations, and where devices compute simple predicates on a small set of possible inputs in their one hop neighborhood (for example, the infection status inf is either zero or one, and the time of infection $tinf$ may be in $[1, 120]$ referring to the days in

the latest few months). We note that although these queries are a sub-class of a broad set of queries, they are important according to the health literature and form a precursor to more sophisticated queries in an analyst's workflow. Besides the limitation on the generality, we don't aim to support interactive queries.

Privacy (P1). Private data should always be hidden from the adversary. From the point of view of a device, it may not learn any information beyond its own node, edge, and topology data. In particular, it may not learn any information about a neighbor beyond what is contained in the direct edge to the neighbor. Similarly, the analyst must only learn the query result.

Privacy (P2). Individual devices will cause limited changes to query result. With privacy goal P1, only the query result would be learnt by the adversary. To prevent adversaries from learning (additional) information from the query result, the system must guarantee that individual devices will make bounded contribution to the result.

Scale and efficiency. First, our system must support a large number of devices, for example, one million. We assume that these devices have a bounded degree, for example, up to 50 neighbors. (If the actual device graph has nodes with a larger degree, then they may select a subset uniformly for the query execution.) Second, in terms of device overhead, we want that to be low, for example, minutes of CPU time and at most a few MiBs in network transfers, which is affordable for mobile devices. Meanwhile, if the system employs any servers, e.g., cloud servers, then we want the server-side cost to be affordable. In terms of dollar cost of renting the servers, we want the per-server cost to be no more than a few tens of dollars (per query over a million devices).

Correctness (non-goal). In this paper, we focus on privacy and require the query output to be correct only during periods when all parties (devices and any servers) are semi-honest. Ensuring correctness in the periods of malicious behavior could be very expensive. Thus we

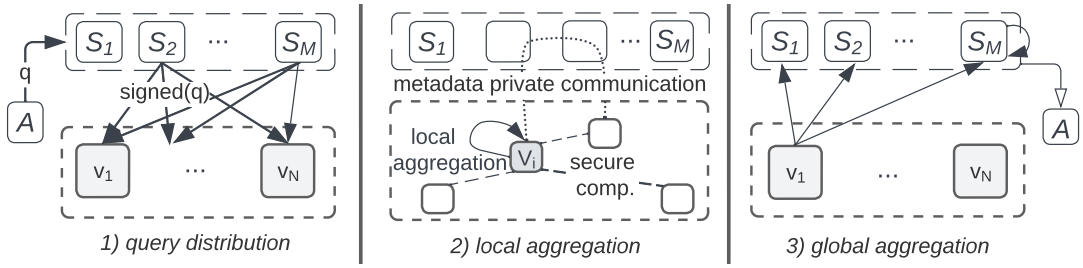


Figure 4.2: An overview of Colo's query distribution, local aggregation, and global aggregation phases of query execution. The dotted line in local aggregation depicts metadata-hiding communication, and the dashed line depicts secure computation.

leave it as future work. Nevertheless, the system must ensure privacy at all times.

4.1.4 Challenge and straw man solutions

Meeting these goals, particularly, privacy and efficiency, when the parties can behave maliciously is hard. We explain this point by discussing possible approaches and attacks below. We will use query Q1 (Figure 4.1), calculating the total number of infected direct neighbors of all infected devices, for illustration purposes.

Centralized server. One approach is to assume a trusted central server and ask devices to upload their data in plaintext. For example, each device uploads the node data (inf) along with its neighbor list, and edge data for all the edges. The server can evaluate the query using a graph analytics system such as GraphX [162] as the data is in a single location in plaintext.

This approach, however, breaks the privacy goal P1 immediately. The server sees individual device data and learns more than the query result.

This approach also breaks the privacy goal P2. Though the devices do not learn more than they should in this approach, malicious devices can execute subtle attacks to cause a victim device contribute more than desired to the query result and thus leak information. For the example query Q1, the server needs to compute $\sum_{v_A,v_B} v_A.inf \cdot v_B.inf$ for all pairs of neighbors (v_A, v_B) to calculate the total number of infections in infected devices' neighbor-

hood. A malicious device v_0 can send a large constant, e.g., $v_0.inf = 10^6$ to the server. As a result, if an honest device v_1 is v_0's neighbor, the product $v_1.inf \cdot v_0.inf$ will contribute a large value in the aggregation and the adversary can infer whether $v_1.inf$ is 0 or 1 by inspecting the query result. As another example of breaking P2, a set of malicious devices can claim that a victim device is their neighbor. Again for the example query, if 10^6 malicious devices claim v_1 is their neighbor and set their inf to be 1, the server will compute $\sum_{v_A,v_B} v_A.inf \cdot v_B.inf + \sum_{10^6} 1 \cdot v_1.inf$. This computation again amplifies the victim device v_1's data in the aggregate output.

To hide the data from the server (privacy goal P1), one approach is to use tools from cryptography such as homomorphic encryption [69, 45] or secret sharing [163, 43]. Meanwhile, to meet privacy goal P2, the server will have to run logic to verify devices' (secret) inputs to prevent malicious devices from uploading arbitrary data. However, this approach doesn't scale to a large number of devices. For example, for Q1, the server will need to perform at least $O(N)$ multiplications (N is the number of devices), and also verifications of devices' data. Indeed, as we discuss in related work (§4.6), we are not aware of any prior work that follows the centralized server approach when the threat model assumes malicious parties.

Federated analytics. Another approach to the problem is the idea of keeping the data federated. Each device keeps its raw data local and first communicates with its neighbors to perform local computation (e.g., aggregation at the neighborhood level), before uploading this local result to a server for aggregation. The federated approach has distinct advantages relative to the centralized approach: (i) it allows devices to feel more trust in the system as they keep possession of their data, (ii) it doesn't risk bulk data breaches as devices do not collectively centralize their data, and (iii) it handles data updates naturally as each device computes locally and can use its latest data. However, efficiency is still challenging.

The state-of-the-art system that follows the federated approach, Mycelium [30], assumes

malicious devices and a single untrusted server and scales to a large number of devices. However, even for our simple example query Q1 and for 1M devices, a Mycelium device spends 8.79 hours in CPU time and 5.73 GiB in network transfers (§4.5). The server is also expensive: it spends 1304 h CPU time and 5737 TiB network transfers. One core challenge is that during the local computation (where a device performs local aggregation over its own and its neighbors' data), Mycelium still needs to protect against the subtle attacks against privacy goal P2 mentioned above (e.g., one malicious neighbor inputting an arbitrary value into the aggregation)—attacks for which Mycelium uses general-purpose (and expensive) homomorphic encryption [164] and zero-knowledge proofs [154]. Besides, Mycelium must protect against arbitrary behavior of the untrusted server, which may try to manipulate the aggregation, e.g., add a device's data a large number of times to learn it.

4.2 Overview of Colo

Given the unique advantages of the federated approach relative to the centralized approach, Colo follows the former. To facilitate federation, Colo relies on a set of servers, for example, 40-100 servers, S_i for $i \in [0, M)$ (Figure 4.2). These servers may be in separate administrative domains (e.g., Azure versus Amazon AWS versus Google Cloud Platform, and different geographical zones within these cloud providers). Colo assumes that at most $f = 20\%$ of these servers may be compromised by the adversary. That is, an adversary may compromise, e.g., up to 8 servers when the number of servers is 40. M and f are both configurable and we followed prior works [148, 149] to use this default setting.

The devices connect to the servers in a star topology, where the central hub of the star is the set of servers and a spoke connects to a device. Thus, the devices do not directly interact with each other; rather, they communicate via the servers. This design is necessary for two reasons. First, the devices typically do not have each other's IP address. Second, the

Setup (one time)

- The servers $S_i, i \in [0, M)$, generate a set of keys for a zero-knowledge proof (ZKP) scheme using a MPC protocol [165].
- The servers distribute the keys to the devices. A device downloads the set of keys from one server and hashes of these keys from the others to defend against attacks from malicious servers (a malicious server can distribute malicious keys). The devices stores the keys locally.

Query distribution

1. The analyst $\mathcal{A}$ submits a query q to all the servers.
2. The servers verify the query and sign it. Each server broadcasts its signed query to all devices to start query execution.

Local aggregation

3. The servers run the Karaoke system [148] to enable the devices to communicate anonymously with each other.
4. Each device initializes its local result of query execution to 0 and runs the local aggregation secure computation protocol (§4.3.3; Figure 4.4) with its neighbors over the Karaoke system [148] to obtain a share of its local result.

Global aggregation

5. Each device secret shares its own share of its local result with the servers.
6. Each server sums all secret shares it receives. Each server sends its sum to the analyst and the analyst who adds the sum across the servers to reconstruct the global result.

Figure 4.3: A high-level description of Colo's phases.

communication via the servers helps hide the devices' topology data.

Colo has a one-time *setup* phase, and three phases of *query distribution*, *local aggregation*, and *global aggregation* for query execution (Figure 4.2).

In the setup phase, the servers generate and distribute keys for a cryptographic protocol used in the local aggregation phase. Specifically, they generate the proving and verification keys for a zero-knowledge proof (ZKP) scheme [165].

In the query distribution phase (the leftmost diagram in Figure 4.2), the analyst $\mathcal{A}$ specifies the query, say q, and sends it to each of the servers. Since $\mathcal{A}$ can be malicious (§4.1.2) and may write a query that tries to infer a single device's data, the servers validate $\mathcal{A}$'s query. For example, if q contains a WHERE clause of the form *WHERE self.ID = xxx*, then the servers

reject it. In general, checking whether a query leaks information about an individual device is an open and difficult research problem [166, 167, 168, 169]. Colo does not aim to solve it and assumes that the servers have a list of certified queries that are allowed. Once each server validates the query, it signs and broadcasts it to the devices. A device starts the next phase after validating enough signatures, that is, more than the threshold of servers that can be compromised.

In the local aggregation phase (the center diagram in Figure 4.2), each device evaluates the query in its neighborhood. For instance, for the query Q1, *SELECT COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf*, each node computes the local count of infected neighbors in its neighborhood. More precisely, each node computes the count of infected neighbor (either zero or one) for each edge, and then adds the results across the edges. Recall that our goal is to ensure that a device learns no more information than it needs to (§4.1.3). Thus, a device uses a secure computation protocol with its neighbor such that at the end of the protocol the two parties receive *secret shares* of the result of the computation. This protocol admits malicious devices; for instance, a malicious neighbor is prevented from supplying an arbitrary input such as setting its $inf = 10^6$.

Secure computation hides node and edge data; however, an adversary that observes network traffic can infer topology by monitoring who is performing secure computation with whom. Thus, the devices in the local aggregation phase execute secure computation over a metadata-hiding communication network, particularly, the Karaoke system [148]. The servers facilitate and run this system.

Finally, in the global aggregation phase (the right diagram in Figure 4.2), the devices send their results from the local aggregation to the servers, who aggregate them. Specifically, each device secret shares its result with the servers, who locally add the shares they receive from the devices. The servers send the result of their local computation to the analyst who combines these outputs across the servers to obtain the query result.

Colo's architecture (with the star topology where the hub is a set of servers) offers advantages for the privacy versus efficiency tradeoff of Colo. First, the servers form a natural infrastructure to instantiate a metadata-hiding communication system such as Karaoke [148, 149]. Second, this architecture keeps the cost of global aggregation low by enabling servers to perform only local computations in the global aggregation phase. In contrast, if we assumed a completely untrusted server as the hub (as in Mycelium [30]), then this untrusted server would have to use a sophisticated verification protocol to prove that it performed the global aggregation correctly (for instance, prove that it did not add a victim device's input many times). There would also be no resourceful infrastructure to run a metadata-hiding communication system.

Nevertheless, there are still several challenges in instantiating this architecture, one of which is the efficiency of the secure computation piece in the local aggregation phase. This protocol has to be particularly efficient in terms of network transfers because this overhead gets exacerbated when secure computation messages are routed through the metadata-hiding network. We next go over the design details of Colo and how it addresses the various challenges.

4.3 Design details

This section describes the details of Colo. Figure 4.3 provides a high-level overview and how the phases connect with each other.

4.3.1 Setup (key generation)

The first challenge Colo must deal with is the overhead of key generation and distribution. As we will describe later (§4.3.3), devices in Colo's local aggregation phase need to generate an array T of values and prove that each value is within a range $[L, U]$. The challenge is that the

efficient and popular zero-knowledge proof (ZKP) schemes such as that of Groth [154] bind the proving and verification keys to the statement (the circuit) the prover is proving. That is, this circuit is a function of the length $len(T)$ of array T in our case. Unfortunately, the $len(T)$ further depends on the analyst's query q, and may be different for different queries.

On the one extreme, Colo could use a single proving and verification key that works for $len(T) = 1$. Then, when proving the properties of an array T with $len(T) > 1$, it could break down the array into singletons and generate a separate proof for each. The benefit is efficiency of key generation and distribution as Colo's servers will need to generate and distribute one key pair. However, the penalty is during query execution as Colo's devices would have to generate many proofs. For instance, for a query with $len(T) = 1024$, the CPU time doubles and the size of the proof increases from 192 B to 192 KiB, relative to generating a single, holistic proof. This network overhead is significant because each message goes through the metadata-hiding communication network (§4.2).

On the other extreme, Colo could generate, say, 1024 keys for all possible lengths of T between 1 and 1024. In this extreme, key generation and distribution is expensive; for instance, the set of keys which has to be shipped to each device (and stored there) will total 52.4 GiB. However, the proof generation is optimal with a single proof for the array T, thereby lowering CPU and network overhead during query execution.

Colo resolves this tension between overhead of key and proof generation by generating a key set $\{key_{2^0}, key_{2^1}, \ldots, key_{2^{\log(len(T))-1}}\}$ containing $\log(len(T))$ keys for all powers of two between 1 and anticipated maximum length $len(T)$, for example, 1024. Now, the size of the key set is manageable, e.g., 102 MiB, and during query execution devices generate no more than $\log(len(T))$ proofs.

4.3.2 Query distribution

As mentioned in the overview (§4.2), the analyst $\mathcal{A}$ specifies a query q and gives it to the servers, who validate it. Here, we elaborate on how $\mathcal{A}$ specifies the query.

Each query has two components: (i) a SQL query similar to the examples we have discussed (Figure 4.1), and (ii) a transformation function, *PreProcess*, that transforms the raw data at a device into a form needed by the SQL query. One can view the *PreProcess* function as implementing preprocessing or cleaning of data and the SQL query as the analysis over the prepossessed data.

As noted earlier (§4.1.3), Colo does not support arbitrary SQL queries. Rather, it targets simple queries of the form *SELECT AGG-OP(g(self.data, neighbor.data)) FROM neigh(1) WHERE h(self.data, neighbor.data)*, where AGG-OP is the SUM, COUNT, or AVG aggregation operation, g is a predicate on the data of two neighbors (including their node and edge data), and h is a filter that runs over the same data to compute whether a particular edge will participate in aggregation or not. If $\mathcal{A}$ wants to run a query with a GROUP BY operation, $\mathcal{A}$ transforms it into several sub-queries and submits them to the servers separately. For example, $\mathcal{A}$ converts Q5 in Figure 4.1 into a series of queries such as *SELECT COUNT(*) FROM neigh(1) WHERE 20 < self.age < 30* for the different age groups.

The *PreProcess* function specifies how a device should translate its raw data into the attributes accessed by the SQL query. For example, the analyst $\mathcal{A}$ may want to execute Q3 in Figure 4.1 (*SELECT SUM(edge.contacts)/COUNT(*) FROM neigh(1) WHERE self.inf & neighbor.inf & (neighbor.tInf > self.tInf+2 days)*) for all infectees during a recent time period, e.g., March 2023, as suggested in the literature [158]. The PreProcess function should specify how to derive the following three attributes for the SQL query: i) device infection status inf as a binary number; ii) the infection timestamp $tInf$ as an integer from 1 to 30 to represent March 1 to 30; iii) $edge.contacts$, which is the number of interactions two neighbors have had, as

a bounded integer, for example, 80 as in the epidemiology literature [35]. These constraints (bounds) are necessary as otherwise malicious devices can supply arbitrary inputs.

Once the servers receive $\mathcal{A}$'s query, they verify it by matching it to a list of certified queries. The servers then sign and broadcast the query; if a device verifies signatures more than the fraction of servers that can be compromised, it starts query execution.

4.3.3 Local aggregation

Hiding node and edge data

Recall from the overview (§4.2) that the goal of local aggregation is to enable a device to compute the query locally just on its neighborhood. This computation further breaks into evaluating the query for every neighbor edge of a node. That is, a node needs to evaluate a function $F = g \circ h(self.data, neighbor.data)$ with each of its neighbors and compute $\sum_{neighbor} F(self.data, neighbor.data)$. As an example, for query Q1 the function F equals $F(self.data, neighbor.data) = self.inf \cdot neighbor.inf$.

For the moment, assume that we do not need to hide the topology data at the devices (we will relax this assumption in the next subsection). Then, a natural option for computing F is to use a two-party secure computation protocol such as Yao's garbled circuit (Yao's GC) [63]. The neighbor, say, v_B, could act as the garbler, generate a garbled circuit, send it to the other node, say, v_A, who would act as the evaluator to obtain the result. Since the two nodes must not obtain the result of F in plaintext, the two nodes may compute $y = F(r_{v_A}, r_{v_B}, v_A.inf, v_B.inf) = (v_A.inf \cdot v_B.inf) + r_{v_A} + r_{v_B}$, where $r_{v_A}, r_{v_B} \in \mathbb{F}$ are uniformly sampled masks supplied by the two parties to hide the output from each other. (The field $\mathbb{F}$ could be 2^{64}, for example.) At the end of the protocol, v_B may store $-r_{v_B}$ as its output, while v_A may store $-r_{v_A} + y$ as its output.

The challenge is in preventing malicious behavior efficiently. To protect against a mali-

Local aggregation protocol of Colo

1. Each device receives a query q containing a function $PreProcess$, a local computation F, and a bound $Bound$. $PreProcess$ contains information on how to convert a device's raw data into inputs for F. It also specifies the set of valid values for each input parameter to F. The bound $Bound$ is the range of valid outputs of F.
2. Each device participates in this local aggregation protocol. Denote A as the device and B as a neighbor.
3. Each neighbor B of A does the following locally:
   (a) (Generate mask) Samples an element r in a field $\mathbb{F}$ uniformly randomly.
   (b) (Enumerate all inputs of A) Generates an array s containing all possible inputs of A based on $PreProcess$.
   (c) (Compute outputs) For every $s[i]$, computes $T[i] = F(s[i], B_{in})$, where B_{in} is B's input.
   (d) (Mask outputs and commit to them) For every $T[i]$, computes $T'[i] = T[i] + r$. It then samples $R[i]$ and generates commitment $CM[i] = Commit(T'[i], R[i])$.
   (e) (Prove bound of outputs) Generates a ZKP that it knows the opening of all commitments and a mask such that each committed value is the addition of the mask and some value bounded by $Bound$.
4. Each neighbor B sends commitments CM and the ZKP to A which verifies the ZKP.
5. (Function evaluation) A runs OT with each neighbor B to retrieve $(T'[j], R[j])$, where $s[j]$ is A's input to F. The device A verifies the opening to the commitment, that is, it verifies $Commit(T'[j], R[j])$ equals $CM[j]$ received in the previous step.
6. If the verifications above pass, A adds $T'[j]$ to its local aggregation result. Its neighbor device B adds $-r$ to its local aggregation result.

Figure 4.4: Colo's local aggregation.

cious neighbor (garbler v_B), Colo would have to use a version of Yao's GC that employs techniques such as cut-and-choose [170, 171, 172] that prevent a garbler from creating arbitrary circuits, for example, an F' that computes, $(v_A.inf \cdot 10^6) + r_{v_A} + r_{v_B}$. These general-purpose primitives are expensive because they are not tailored for the queries and they offer more than we need: malicious integrity is not our goal (§4.1.3).

Observe that the predicates F that appear in Colo's target queries have bounded inputs and outputs. For example, Q1 has two possible inputs of 0 and 1 for $v_A.inf$ and thus two possible outputs of 0 and 1. As another example, if we take Q3 in Figure 4.1 and assume $tInf$

has 30 possibilities (for 30 days), then the number of possible inputs for v_A is sixty. That is, v_A's input pair $(inf, tInf)$ can range from the case $(0, 0)$ to the case $(1, 29)$. Corresponding to each of these inputs, an output $edge.contacts$ may be a value in the range $[0, 79]$.

Leveraging this observation, the neighbor in Colo (v_B above) *precomputes* outputs for all possible inputs of v_A into an array T instead of generating them at *runtime* inside Yao's GC. One can view this arrangement as making v_B commit to the outputs without looking at v'_As input. For instance, for Q3, v_B would generate a T of length 60 where each entry is in the range $[0, 79]$. Then, v_B can arrange for v_A to get one of these values obliviously.

Protocol details. Figure 4.4 shows the details of Colo's protocol. For a moment, assume that the nodes v_A and v_B are honest-but-curious. Then, for every possible input of v_A, the neighbor v_B generates one entry of array T. It also adds a mask, $r \in \mathbb{F}$, to each entry of T. That is, after generating the array T, v_B offsets each entry by the same mask and computes $T'[i] = T[i]+r, i \in [0, len(T))$, where r is private to v_B. The node v_A then obtains one of the entries of the table corresponding to its input using 1-out-of-$len(T)$ oblivious transfer [44, 151]. Finally, v_A uploads $T[j] + r$ for the global aggregation, and v_B uploads $-r$ to cancel the mask.

To account for a malicious neighbor v_B who tries to break our privacy goal P2 (§4.1.3), Colo's protocol adds a zero-knowledge range proof [154]. Specifically, v_B commits to the values in T and proves that each value is bounded and that each value is offset by the same private mask r. Recall from the setup step (§4.3.1) that the keys for the ZKP are specific to length $len(T)$ of array T. Thus, v_B splits up proof generation as needed depending on the binary representation of $len(T)$. After generating the proofs, v_B sends them to v_A along with one entry $T'[j]$ of T' (using OT) as well as the opening for the commitment to $T'[j]$. Now, instead of using an OT protocol secure only against an honest-but-curious sender, Colo's protocol switches to an OT that defends against a malicious sender (v_B) [44, 173].

Colo's protocol is not general purpose and is not meant to replace Yao's GC. In particular,

its overhead is proportional to the input domain size of the predicate F. Thus, it only works for small input domains. However, its performance benefits are very significant, and allows the devices to keep their overhead low. First, node v_B evaluates the predicate F in plaintext rather than inside Yao's GC. Second, the protocol uses ZKP for range proofs, a computation which has received significant attention in the literature [174, 175]. Third, and similar to the point above, the OT primitive is also a basic secure computation primitive that has received much attention on performance optimizations [44, 173, 176, 177].

Hiding topology

The local aggregation protocol so far hides node and edge data; however, an adversary that can monitor network traffic can infer who is running secure computation with whom and thus figure out the topology data for the devices. As noted earlier (§4.2), Colo protects this information by enabling the devices to communicate over a metadata-hiding communication system. In particular, Colo uses a state-of-the-art system called Karaoke [148]. Here, we briefly review Karaoke as it has significant implications on Colo's server-side overhead.

At a high level, Karaoke relies on the concept of *dead drops*—a pseudorandom location (mailbox) at a server. Say device v_A wants to send a message to v_B, and they have a shared secret k_{AB} to facilitate this communication (e.g., they can agree on this secret when they become neighbors). To send a message msg to v_B, the device v_A first derives a dead drop and picks a server S_{drop} hosting the dead drop, using k_{AB} as the seed of a pseudorandom generator [178, 179, 180]. The device v_A then drops (writes) the message at the dead drop, while device v_B retrieves the message from the dead drop.

Of course, the idea of dead drops alone is insufficient, as the dead drop server S_{drop} can see which devices are communicating with the same dead drop. To fix this issue, Karaoke makes the devices access the dead drops via servers, similar to a parallel mixnet [181]. Suppose the key pairs of two devices v_A, v_B are (pk_A, sk_A), (pk_B, sk_B), and the key pairs for

the M servers $S_i, i \in [0, M)$'s are (pk_i, sk_i). Then, when the device v_A wants to send a message to a dead drop server, it chooses a uniformly random path of m servers, say, $S_0, \ldots, S_{m-1}$, to route the message. Specifically, v_A onion encrypts [181] the message as $Enc_{pk_0}(\ldots Enc_{pk_{m-1}}(Enc_{pk_{drop}}(msg)))$ and sends it to the first hop on the path. When a server on the path receives a message, it peels one layer of the onion and forwards the remaining message to the next server, until the drop server receives it. To protect against malicious servers dropping messages to compromise privacy, the servers add dummy traffic (noise messages to random dead drops).

Karaoke adds both significant CPU and network overhead to the servers (§4.5).

Despite this overhead, we pick Karaoke for several reasons. First, the overhead is at the server-side, where there is more tolerance for overhead relative to the devices. Second, Karaoke can scale to millions of devices as needed for our scenario. Third, it defends against malicious servers who may duplicate or drop traffic to learn access patterns at the dead drops. And, fourth, it provides a rigorous guarantee of (ϵ, δ) differential privacy. For instance, it makes the case that v_A and v_B are communicating appear indistinguishable from the case when they are idle. In particular, with a route length $m = 14$, assuming 80% of the servers are honest, it guarantees indistinguishability with a probability of $1 - 2 \cdot 10^{-8}$.

4.3.4 Global aggregation

The global aggregation phase is the last phase in query execution. Recall that at the end of the local aggregation phase, each device has a local result, say y_i. For instance, y_i equals the output of secure computation $T'[j]$ in Figure 4.4 or the mask r added by a device that constructed the array T'. The goal of the global aggregation phase is to aggregate these outputs across devices to wrap up the execution of the query. For this aggregation, each device secret shares (using additive shares in the field $\mathbb{F}$) its output y_i and sends one share to

each server. Each server then locally computes $\sum_i [y_i]$, where $[y_i]$ is a share it receives. The analyst finally aggregates the results across the servers, that is, computes $\sum_{edge}(T[j] + r) + \sum_{edge}(-r)$, canceling out the random masks, and obtaining the query output.

This simple global aggregation protocol ensures that the local aggregation results of an honest device get aggregated with the results of other honest devices exactly once. For instance, if a malicious server drops or duplicates a share $[y_i]$ supplied by an honest device, then the analyst $\mathcal{A}$ learns a uniformly random output, because the shares of $[y_i]$ at an honest server will not cancel out. Naturally, this protocol guarantees privacy as long as one of the servers is honest, which is satisfied by our assumption of only up to $f = 20\%$ of servers being malicious (and the rest are honest).

4.4 Implementation

We have implemented a prototype of Colo in C++. Our prototype is approximately 2,000 lines of code on top of existing frameworks and libraries. In particular, we use CAF [182], an event-driven framework used commonly for building distributed systems. We instantiate Colo's servers and devices as actors in this framework. The source code will be made available on Github.

Server-side details. We use Groth16 [154] as our underlying zkSNARK (ZKP) scheme and we use BLS12-381 as its underlying curve for a 128-bit security. In the setup phase, our prototype generates a set of ZKP keys in MPC using ZoKrates [165]. This protocol ensures that the randomness used to generate the keys is not revealed to an adversary, and that the generated keys are correct. In the local aggregation phase, servers run Karaoke [148] to provide metadata-hiding messaging. We follow Karaoke's default configuration to generate noise messages to achieve $\epsilon = \ln 4, \delta = 10^{-4}$ differential privacy after 245 rounds of message losses. In the global aggregation, servers accept devices' additive secret shares of local results and reveal

Application	Dataset	# of nodes
Social	ego-Facebook	4,039
Communication	Enron email	36,692
Collaboration	DBLP	317,080
Collaboration	live journal	3, 997,962

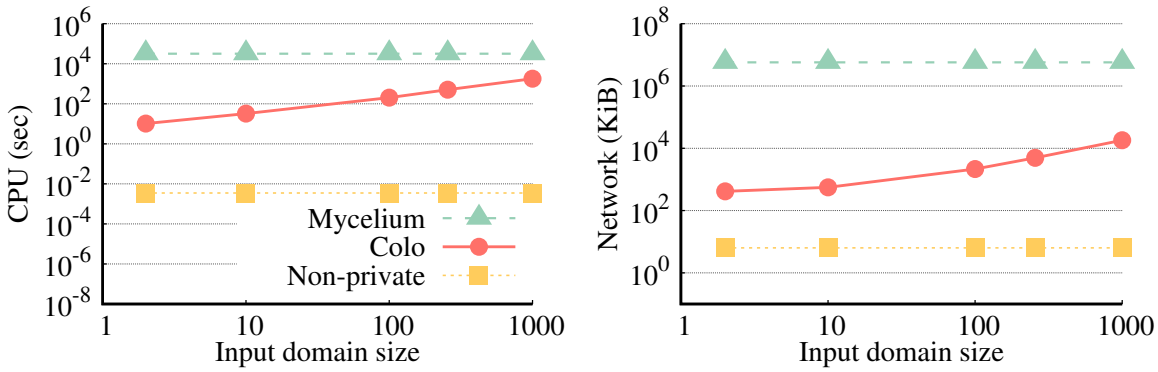
Figure 4.5: Commonly used graph datasets from the Stanford Large Network Dataset Collection (SNAP) [186].

the summation of these results to the analyst. Our prototype uses 2^{64} as the underlying field $\mathbb{F}$ for secret shares.

Device-side details. In the query distribution phase, the devices take as input two functions in C++, one for the predicate F and the other for the *PreProcess* transformation function (§4.3.2). The devices also take as input a map object indicating the range of each attribute accessed by the query. For the local aggregation phase, recall (Figure 4.4) that devices need cryptographic tools, particularly, a commitment scheme, a ZKP scheme, and an OT protocol. For commitments, our prototype uses a ZKP-friendly scheme called Poseidon [153], by importing the ZKP circuit for this commitment from neptune [183] into ark_groth16 [90]. It also implements a range proof to prove that the values in the array T are in the range bound. For OT, we use the simplestOT protocol [44] implemented in libOTe [184]. For enabling the devices to participate in Karaoke (metadata-hiding communication), we use the onion encryption scheme implemented in libsodium [185]. Finally, for global aggregation, the devices simply sample random numbers in 2^{64} to generate the additive secret shares of their local results.

4.5 Evaluation

Our evaluation focuses on highlighting the device- and server-side overhead of Colo, for a variety of queries, and for different underlying graphs (the number of nodes and edges, and the degree of these nodes). A summary of our main results is as follows:



(a) Device CPU (b) Device CPU

Figure 4.6: Device-side cost with a varying input domain size $len(T)$ of the query predicate. Queries Q1, Q2, Q4, Q7 from Figure 4.1 have $len(T) = 2$, Q3, Q5, Q6 have $len(T) \in [60, 240]$, and Q8 has $len(T) = 240$.

- For 1M devices connected to up to 50 neighbors each, and for all example queries (Figure 4.1), Colo's per-device cost is less than 8.4 min of (single core) CPU time and 4.93 MiB of network transfers.
- For the same number of devices and their neighbors as above, Colo's server-side cost is $3.95 to $37.6 per server (or $158 to $1,504 total for the 40 servers), depending on the query.
- Colo's overheads for the example queries are much lower than the state-of-the-art Mycelium, whose device-side cost is 8.79 hours (single core) CPU time and 5.73 GiB network transfers, per query, and the server-side cost is $\$57,490$ per query. Thus, Colo brings privacy-preserving federated graph analytics into the realm of affordability for certain types of queries.

Baselines. We compare Colo to two baseline systems: a federated non-private baseline and the state-of-the-art Mycelium [30] for privacy-preserving federated graph analytics.

Our non-private baseline has a coordinator server. It forwards messages between devices as they typically do not have public IP and cannot interact directly. Using the coordinator,

devices run graph queries in a federated manner (but they are not privacy-preserving): each neighbor v_B of v_A sends its data to v_A in plaintext, who then performs local computation and uploads the result to the coordinator. The coordinator aggregates the results across devices. This baseline is non-private as devices see their neighbors' data, and the coordinator sees the local results computed at the devices. Besides, this baseline does not consider malicious devices.

For a privacy-preserving baseline, we use the state-of-the-art Mycelium [30]. However, this is challenging as Mycelium only has a partial public implementation [187]. In particular, the device-side zero-knowledge proof implementation does not include the full proof, and the server-side code does not include the aggregation of computation across devices. To work around these constraints, we report lower-bound estimates for Mycelium. Briefly, however, for device-side CPU, we use a lower-bound estimate of ZKP since it is the dominant device-side cost (Mycelium performs homomorphic operations over ciphertexts and proves that these operations are correct). For the server-side costs and the network overhead, we use a mix of the released code combined with reported numbers from their paper. For details, we refer readers to Appendix B.2.

We emphasize that Mycelium has a different and a stronger threat model than Colo—Mycelium assumes a single byzantine server, while Colo assumes a set of servers (e.g., 40) of which 20% can be byzantine. Thus, Mycelium is not a direct comparison (but the closest in the literature), and we use it to situate Colo's costs, as both Mycelium and Colo operate in strong threat models.

Queries and datasets. We measure and report the overhead for the example queries in Figure 4.1. Recall that Colo's overhead depends on the size of the input domain of the query predicate, that is, the length $len(T)$ of array T which contains a value for every possible input to the predicate. Thus, we vary $len(T)$. For example, if we assume inf has 2 possible

values, $tInf$ is any value between 30 and 120 [157, 158, 160, 161], and age has 120 possibilities, the queries Q1, Q2, Q4, Q7 have $len(T) = 2$, Q3, Q5, Q6 have $len(T) \in [60, 240]$, and Q8 has $len(T) = 240$. Given these numbers, we vary $len(T)$ between 2 and 1000, to cover our example queries.

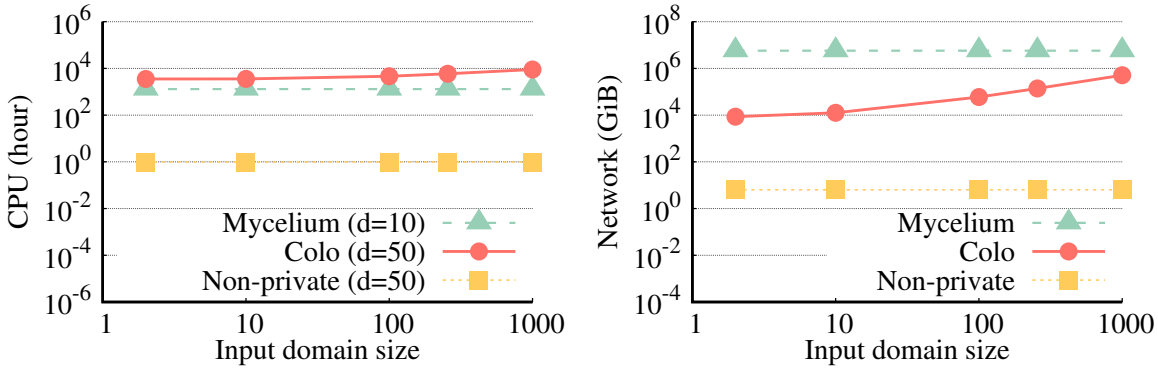
For the number of nodes and edges in the underlying graph, we take inspiration from several public datasets from the Stanford Large Network Dataset Collection (SNAP) [186] (Figure 4.5). In terms of topology, we vary the maximum degree of nodes between 1 and 100; setting this maximum degree is necessary, as each device must communicate with a fixed number of nodes (with itself if it does not have enough neighbors) to hide its topology.

Testbed & methodology. Our testbed is Amazon EC2 and we use machines of type c5.4xlarge to run the devices and the servers. Each such machine has 16 cores, 32 GiB memory, and costs $0.328 per hour [188]. For experiements, we don't run all devices, e.g., 1M, in one go, which would require thousands of machines. Instead, we sequentially run batches of 1K devices each. In a batch, each device connects to e.g. 50 neighbors. This is acceptable as a device's overhead only depends on its neighborhood. Servers have a fixed-cost (dummy/noise messages; described in § 4.5.3) for 1M devices, plus incremental costs per-device. We measure and aggregate these separately.

The one-time setup phase cost is not included in our experiments because our focus is the per-query cost. For completeness, we estimate this cost: the server-side cost is in the order of minutes of CPU time and a few GiBs in network transfers [165, 189] and the device-side cost is 102 MiB (§4.3.1).

4.5.1 Overhead for different queries

This section evaluates and compares the overhead of Colo and the baselines for different queries, that is, different sizes of input domain $len(T)$ of the query predicate. More precisely,



(a) Server CPU (b) Server network

Figure 4.7: Total server-side cost for Colo (across its 40 servers combined) and the baselines with a varying input domain size $len(T)$ of the query predicate. The total number of devices is 1M and the degree of each is 50 (except for Mycelium's CPU cost where it is 10).

we fix the query predicate to $F(v_A.input, v_B.input) = v_A.input \cdot v_B.input$, that is, the predicate for query Q1, and set different input sets for $v_A.input$. For instance, $v_A.input \in [0, 1]$ fixes $len(T) = 2$, $v_A.input \in [0, 9]$ fixes $len(T) = 10$, and so on. For these experiments, we fix the number of devices to 1M where each device has at most 50 neighbors.

Device-side overhead. Figure 4.6 shows per-device CPU and network overhead as a function of $len(T)$. Colo's overhead for $len(T) = 2$ (i.e., query Q1, Q2, Q4, and Q7 in Figure 4.1) is 10.27 s and 415 KiB, and increases to 8.42 min and 4.93 MiB for $len(T) = 256$ (this covers all queries in Figure 4.1), and 29.9 min and 18.12 MiB for $len(T) = 1000$.

Colo incurs significantly more overhead than the non-private baseline. For instance, for $len(T) = 256$, the baseline's overhead is 0.003 s and 6.36 KiB per device, while Colo's overhead is $1.68 \cdot 10^5$ and 775 times higher (505.1 s and 4.93 MiB) for the CPU and the network, respectively. This is because the non-private baseline does the involved cryptographic operations, and devices simply share their data with their neighbors, who perform local computations in plaintext. In contrast, in Colo a device's overhead is dominated by the cost of running the local aggregation protocol (§4.3.3) that involves primitives such as commitments, ZKP, and OT (Figure 4.4).

However, relative to Mycelium, Colo's overhead are significantly lower. For instance, for $len(T) = 256$, a device in Mycelium requires 31, 650 s CPU time (8.79 h) and 5.73 GiB network transfers, while Colo's 505.1 s and 4.93 MiB is $62.6\times$ and $1.16 \cdot 10^3\times$ lower. Mycelium's costs are higher because of various reasons. First, it performs local aggregation using homomorphic encryption and ZKP. For instance, devices multiply ciphertexts containing the inputs of the neighbors, and prove that this multiplication is correct using a ZKP (that a malicious device did not manipulate the computation to reveal another device's input). Second, the homomorphic encryption ciphertexts are large (e.g., 4.3 MiB) as Mycelium must choose large parameters for homomorphic encryption to support ciphertext-ciphertext multiplications. This means that the circuit to do the proof is also large, thereby requiring hundreds of seconds of proving time per ciphertext multiplication. Third, a Mycelium device must participate in a verification protocol for global aggregation (besides participating in local aggregation) to check the byzantine aggregator's work. And, fourth, Mycelium also instantiates a mixnet over the devices to hide their topology data, again contributing to the device overhead. In contrast to Mycelium, Colo's cost is dominated by local aggregation only as the metatdata-hiding communication is instantiated over the resourceful servers, and the global aggregation is also lightweight (§4.2, §4.3.3, §4.3.4). Further, the dominant local aggregation requires ZKP-friendly commitments and range proofs alongside OT, primitives that have been optimized in the literature (§4.3.3).

Although Colo's overheads are lower, they increase with $len(T)$ as the number of cryptographic operations in Colo's local aggregation protocol depend linearly on $len(T)$. For instance, the CPU time of a Colo device is dominated by the time to generate the ZKP—that each entry of T is within a range. This cost depends on the number of entries, with the proof for a single entry taking 0.04 s. Similarly, a Colo device's network overhead increases approximately linearly with $len(T)$; for instance, a device sends one commitment per entry of T. In contrast, the overheads for the baselines do not depend on $len(T)$ as their computation

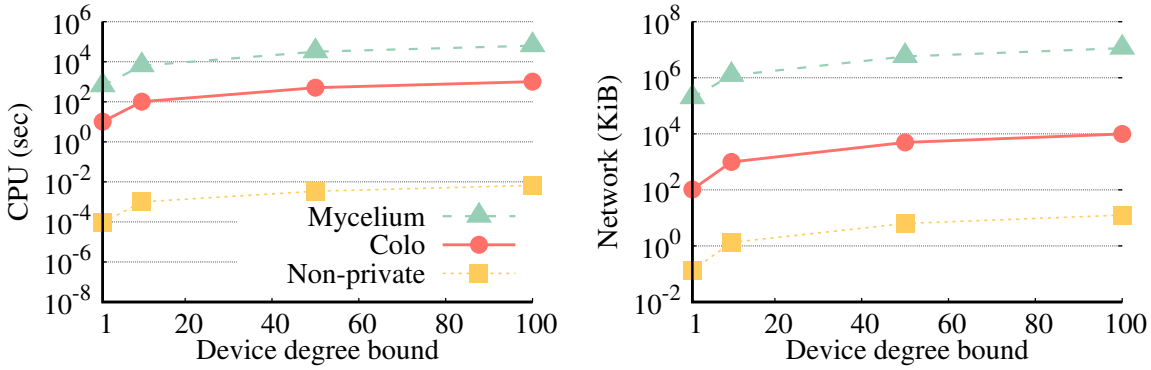
model multiplies $v_A.input \cdot v_B.input$ directly, rather than enumerating all possible outputs. Thus, Colo's overheads will surpass that of Mycelium for a large $len(T)$.

Overall, for Colo's target queries, its device-side overheads (in the range of a few seconds to a few minutes in CPU time, and a few hundred KiBs to a few MiBs in network transfers) appear to be in the realm of practicality for modern mobile devices.

Server-side overhead. Figure 4.7 shows the server-side overhead for Colo and the baseline systems. Colo's server-side cost for $len(T) = 2$ is 88 h CPU time (on a single core) and 214.78 GiB per server, increases to 147.36 h and 3.46 TiB for $len(T) = 256$, and 224.17 h and 12.79 TiB for $len(T) = 1000$.

As with the device-side overhead, Colo's costs are significantly higher than the non-private baseline, whose server incurs 58.6 min CPU time and 6.36 GiB network transfers. The baseline server only needs to forward 32-bit messages from devices to other devices; in contrast, Colo's servers are responsible for hiding topology. They not only forward onion-encrypted messages from the devices over $m = 28$ hops, but also generate noise messages as part of the Karaoke system to protect against malicious servers (§4.3.3).

Relative to Mycelium, Colo's server-side CPU is comparable, while the network is significantly lower. (For Mycelium's server-side CPU, since we take numbers from their paper we are only able to report for a degree bound of 10, which is what they experiment with.) For instance, for $len(T) = 256$, the server-side cost for Mycelium is 1304 h and 5737 TiB, while it is 5894 h and 138.4 TiB for Colo's 40 servers combined. The CPU cost for Mycelium is dominated by the cost to verify ZKPs (for homomorphic operations) and perform global aggregation (over its byzantine server), while the CPU cost for Colo is dominated by the time to process onion-encrypted messages. Both overhead turn out to be comparable. Meanwhile, the network overhead due to these dominant operations is lower in Colo. Although Colo's server-side costs are substantial, since servers are resourceful (e.g., the CPU time can be split



(a) Device CPU (b) Device network

Figure 4.8: Device-side costs with a varying node degree. These experiments set $N = 1M$ devices and input domain $len(T) = 256$ for the query.

across many cores), we consider it affordable. We further assess this affordability by calculating the dollar cost of renting the servers.

We take both the CPU and network cost and convert it into a dollar cost per query using the following pricing model. Each c5.4xlarge machine on AWS costs $0.328 per hour and has 16 cores and 32 GiB memory [188]. We assume that all hourly cost is due to the CPU, and compute the CPU cost per hour to be $0.0205. For the network cost, we set the cost of transferring one GiB to $0.01 according to the typical bulk network pricing of cloud providers [190, 191].

Applying this pricing model, Colo's dollar cost per query (over 1M devices) ranges from $3.95 to $37.6 per server, or $158 to $1504 total for the 40 servers, depending on the query. In contrast, the non-private baseline costs $0.08 and Mycelium costs $57,490 per query. These figures for Colo are substantial but within the reach of the budget of an entity like CDC. For instance, running the cheapest query (which includes the superspreader query) every two weeks will cost less than $4K annually.

4.5.2 Overhead with the degree of devices

In this subsection, we evaluate how the device-side costs change with the degree bound (the maximum degree of a node). Recall that setting a maximum degree is necessary, as each device must communicate with a fixed number of nodes (with itself if it does not have enough neighbors) to hide the communication pattern.

Figure 4.8 shows the device-side CPU and network costs for the three systems for degree bounds of 1, 10, 50, and 100, for $N = 1$M devices and $len(T) = 256$. We report only the device-side costs here because the server-side costs change with the total number of devices, and increasing the degree bound is equivalent to increasing the number of devices from a server's point of view (§4.5.3).

When the degree bound is 10, Colo's devices spend 1.68 min in CPU and 989.77 KiB in network transfers. This cost increases to 16.8 min CPU and 9.85 MiB network with a degree bound of 100. The costs increase and decrease with the degree bound because each device participates in one instance of secure computation (Figure 4.4) per neighbor. This linear dependence is present even for the baselines. For instance, Mycelium's per-device costs are 1.75 h and 1.16 GiB for a degree bound of 10, and 17.5 h and 11.16 GiB for a degree bound of 100. Overall, Colo can scale to a significant degree bound (e.g., 10 or 50) with affordable costs for the devices.

4.5.3 Overhead with the number of devices

This subsection evaluates how the three systems scale with the total number of devices. Specifically, we set the number of devices (nodes) from our example datasets (Figure 4.5), that is, $N \in \{4K, 36.6K, 317K, 3.99M\}$. We also consider $N = 1M$. We fix $len(T) = 256$ and the maximum degree of a node to 10 in alignment with Mycelium's default setting for a better comparison to its server-side CPU costs (§4.5.1). Further, since the device-side costs do not

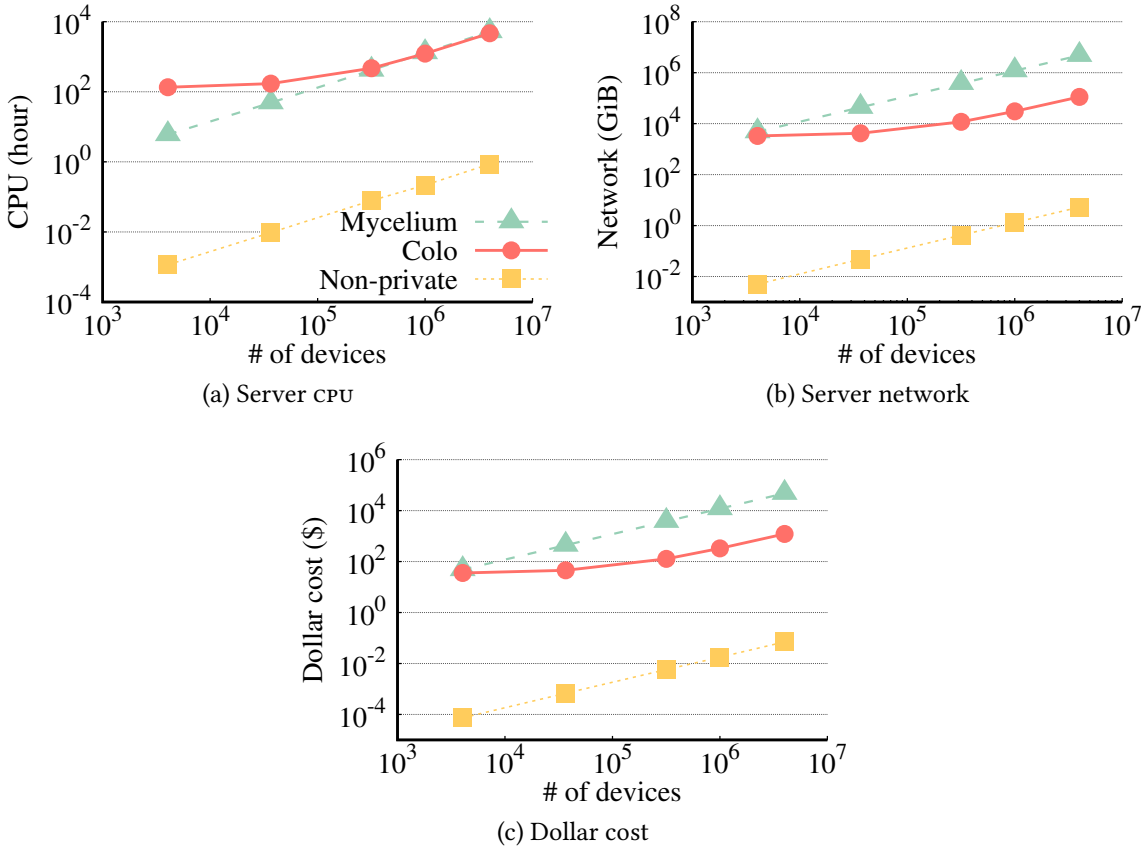


Figure 4.9: Total server-side cost for Colo (across its 40 servers combined) and the baselines with a varying number of devices N. The degree of each device is 10 and the input domain size of the query predicate is set to $len(T) = 256$.

depend on the number of devices, rather just the configuration of their local neighborhood, we focus on the server-side costs.

Figure 4.9 shows these costs, that is, CPU time, network transfers, and dollar cost for the servers. Colo's cost is 3.37 h CPU time, 82.15 GiB network transfers, and 0.89 dollars, per server, for $N = 4039$ devices, increases to 30.74 h CPU time, 758.09 GiB network transfers, and 8.21 dollars per server for $N = 1M$ devices, and further increases to 116.67 h CPU time, 2.79 TiB network transfers, and 30.29 dollars per server for $N = 3.99M$ devices.

The server-side costs for Colo (and the baselines) increases with the number of devices. This is expected as the servers do more work for more devices: send more onion-encrypted

messages in the case of Colo, route more plaintext messages in the case of the non-private baseline, and operate over more homomorphic encryption ciphertexts (e.g., in global aggregation) for Mycelium. However, Colo's cost is flatter and higher for a lower N. This is because even for a small number of devices, Colo's servers must generate and process several million noise messages (§4.3.3) as required by Karaoke to protect against malicious servers dropping traffic to learn access patterns to dead drops. For instance, for $N = 4039$ devices, 3.25 h (96.4%) and 79.41 GiB (96.7%) of the server-side cost in Colo comes from the noise messages. However, this fixed cost becomes less significant as the number of devices increases, thus bringing Colo's costs lower than Mycelium. For instance, Colo incurs 35.6 dollars in total for its 40 servers relative to 48 dollars in Mycelium ($1.34\times$) for $N = 4K$, but 1211.6 dollars versus 47,690 dollars for Mycelium ($39.3\times$) when N equals $3.99M$.

Overall, if we consider the dollar cost per server as a key metric for Colo's costs, then Colo appears to scale to several million devices. However, scaling it beyond these values would require further optimizations and refinements at its servers.

4.6 Related work

4.6.1 Graph analytics with a single data owner

A long line of research in graph analytics [192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206] focuses on outsourcing where a single data owner outsources computations over its graph to one untrusted server or a set of non-colluding servers. One common goal for these works is to hide the graph data from the server(s). For this purpose, they use different cryptographic tools or assumptions.

Flare [192], for instance, outsources computation to servers with a trusted execution environment (TEE), e.g., Intel SGX. Specifically, it runs Spark-like computation inside TEE, which

can support the GraphX interface. Other systems [193, 194, 195, 196, 197, 198] use homomorphic encryption instead of TEEs to run a variety of graph queries on an encrypted graph, including search [195, 196] and shortest distance computations [197, 198]. In the multiple-server setting, a common technique is to use secret sharing and multiparty computation to hide the graph from the servers [199, 200, 201, 202, 203, 204, 205]. For instance, GraphSC [204] assumes two honest-but-curious servers and supports parallel execution of secure computation for a broad class of tasks. While GraphSC targets honest-but-curious servers, MAGO [206], which is designed for subgraph counting, assumes three servers where one can be malicious.

The biggest difference between Colo and these existing works is the problem setting. While these prior works target a single data owner who outsources computation, Colo targets a setting with many data owners (e.g., a million devices). Naturally, in Colo's setting it is natural to consider the threat of malicious devices. This difference in setting and threat model leads to a vastly different system architecture (§4.2) and protocol design (§4.3).

4.6.2 Federated graph analytics

Private data federation [207, 208, 209, 210, 211, 212] focuses on a scenario where a set of data owners hold private relational data and a query coordinator orchestrates SQL queries on this data. The goal is to keep the sensitive data of the data owners private. Private data federation has two differences with Colo. First, they natively target the relational data model while Colo targets the graph data model. (We include these works here because sometimes graph queries can be expressed in the relational model.) Second, these existing works target a scenario with a few data owners (e.g., a few tens) where each holds a significant partition of the relational data. For instance, Senate [212] experiments with 16 parties. In contrast, Colo targets millions of participants where each has a small amount of data. Third, while most works in private data federation consider parties to be honest-but-curious, it is natural

in Colo to consider malicious parties given their number.

DStress [213] focuses on graph queries and a larger number of participants: a few thousand. The key use case is understanding systemic risk for financial institutions, which requires analyzing inter-dependencies across institutions. However, DStress also assumes honest-but-curious participants as financial institutions are heavily regulated and audited, and thus unlikely to be malicious.

Gunther et al. [147, 146] consider malicious devices alongside a set of semi-honest servers to answer epidemiological queries such as an estimate of the change in the number infections if schools are closed for 14 days. However, they are limited to secure aggregation across neighbors' data and their protocol does not hide the result of local aggregation.

Mycelium [30] is the closest related work to Colo. It supports a broad set of queries, targets a large number of devices and assumes they can be malicious. However, as discussed earlier (§4.1.4) and evaluated empirically (§4.5), its costs are very high. Colo is a more affordable alternative in a strong threat model, at least for the subset of queries that Colo targets.

4.7 Summary

Privacy-preserving federated graph analytics is important as graphs are natural in many contexts. It specifically is appealing because it keeps raw data at the devices (without centralizing) and does not release any intermediate computation result except for the final query result. However, the state-of-the-art prior work for this problem is expensive, especially for the devices that have constrained resources. We presented Colo, a new system that operates in a strong threat model while considering malicious devices. Colo addresses the challenge of gaining on efficiency through a new secure computation protocol that allows devices to compute privately and efficiently with their neighbors while hiding node, edge, and topology data (§4.3.3, §4.3.3, §4.2). The per-device overhead in Colo is a few minutes of CPU and a few

MiB in network transfers, while the server's overhead ranges from several dollars to tens of dollars per server, per query (§4.5). Our conclusion is that Colo brings privacy-preserving federated graph analytics into the realm of practicality for a certain class of queries.

Chapter 5

Virtual Pooling

In this chapter, we describe an efficient system for our third type of target queries, privacy-preserving federated statistical queries among multiple health systems.

Nowadays the rapid growth of biomedical data coupled with impressive advances in data science have revolutionized biomedical research at all scales, from molecules to populations. The application of analytic methods to biomedical "big data" has illuminated fundamental insights into the basis of human health and disease, and has informed strategies on disease treatment and prevention. For example, a recent multi-site collaboration led by the National Institutes of Health (NIH), the National Covid Cohort Collaborative (N3C), pooled electronic health records (EHR) data from across the US to facilitate COVID-19 research, and has led to many high-impact papers, such as one study [2] on the association between chronic immunosuppressive use and in-hospital outcomes of patients with COVID-19.

The traditional approach, pooling data, can be slow-paced, as the clinical researcher has to sign a data sharing contract with each site, before the site extracts and sends data. Signing of a data sharing contract can take months or even years. Some sites may not even be willing to participate due to either technical burden or perceived data security risks. A single data breach of the server hosting the pooled data repository can compromise the data of millions

of patients. Even if patient-level patient records are de-identified, there is inherent risk that an adversary may reidentify individuals.

Another approach for large-scale, multicenter CER over networks of EHR data is distributed querying. A coordinating center or study lead distributes queries through secure channels and gets back aggregate results (e.g., summary statistics) from each site. The researcher then combines the aggregate results into a single set of results, for example, using meta-analysis. However, with this approach, many statistical regression models cannot be learned precisely.

Inspired by the area of federated learning in computer science, we recently developed a new privacy-preserving method called virtual pooling [41] that combines the best elements of these current methods. It allows clinical researchers to analyze data as if they were pooled in a centralized repository, but with the patient-level data remaining secure at each center. Virtual pooling adds layers of security and privacy tailored for protected health information, using an advanced encryption technology. Although the conceptual framework underlying virtual pooling has already been developed (US patent application 18/493,571), its utility in healthcare research settings has not yet been tested.

Contributions:

- To the best of our knowledge, virtual pooling is the first privacy-preserving federated system that supports an end-to-end statistical analysis pipeline and gives the same accuracy as the centralized baseline.
- At the core of virtual pooling is secure and optimized computation of a set of statistical methods, including descriptive statistics, propensity score matching and Cox PH models, Kaplan-Meier models.
- A prototype implementation and an experimental evaluation of virtual pooling

5.1 Problem statement

5.1.1 Scenario

We consider a real-world scenario consisting of multiple health systems and an analyst. The health systems contains EHR data of many patients, for example, age, sex, use of different medicines, in hospital days, death and etc. This EHR data is extremely sensitive and the health systems are not allowed to share the raw data with other parties.

The analyst wants to use some statistical methods to analyze the data across multiple health systems. For example, the analyst wants to learn whether a medicine has some effect for some disease. The primary reason the analyst seeks to analyze data from multiple health systems is to ensure high data quality. More health systems mean larger dataset. Also, even if one health system has enough number of samples, there might be some bias among those patients since they are in the same region.

We are motivated by a high-impact study [2] that focuses on the association between chronic immunosuppressive use and in-hospital outcomes of patients with COVID-19 and our goal is to replicate the study with virtual pooling.

5.1.2 Threat model

In our target study, there are only 59 health systems and we assume each health system is honest-but-curious. Besides, we assume no more than half of them will collude with each other. We also assume that there is a server that will talk to the analyst and coordinate between health systems. Similarly, we assume the server is honest-but-curious.

In biomedical community, it's considered secure to reveal aggregate-level information, e.g., average of ages across patients from some health system. So in virtual pooling, we also consider the leakage of summary-level information secure.

5.1.3 Goals

Privacy. It must guarantee that no patient-level information will be leaked under our honest-but-curious threat model.

Functionality. With virtual pooling, we should be able to re-run the analysis from our target study [2], which we will describe in details later. The results virtual pooling produce should be identical to those given by the centralized baseline. We don't want to lose accuracy here because as we have seen from approximation methods, accuracy loss is usually hard to bound. However, if the accuracy loss cannot be bound, then it's hard for the analyst to trust the results.

Efficiency. For efficiency we focus on latency as well as communication rounds. The reason is health systems usually have plenty of computation power and bandwidth. Instead, the latency affects the analyst' efficiency.

5.1.4 Centralized analysis

The centralized analysis [2] have three steps: preprocessing, propensity score matching and Cox propotional hazard models. The raw data at each health systems are multiple relational tables and the preprocessing takes tables as input and outputs one table that consists of the patients and covariates that we are interested in. This step can happen locally at each health system.

Propensity score matching. Propensity Score Matching (PSM) is a statistical technique used in observational studies to reduce selection bias. PSM usually consist of two steps:

- Calculate propensity scores. The first step in PSM is to calculate each individual's propensity score, which is the probability of receiving the treatment based on observed covariates. In our target study, logistic regression is used to compute the scores.

- Match based on scores. After calculating the propensity scores, treated individuals (those who received the intervention) are matched with untreated individuals (those who did not) who have similar scores. In our target study, nearest neighbor matching is used.

Since the target analysis uses iteratively reweighted least squares method to train logistic regression model, which is equivalently to Newton-Raphson, we give an overview of how to train logistic regression model using Newton-Raphson.

Suppose the data is (X, y), where X has N rows and p features and y has N elements of labels. The log likelihood is defined as

$$l(\beta) = \sum_i y_i \log(\sigma(\beta^T x_i)) + (1 - y_i) \log(1 - \sigma(\beta^T x_i)), where\ \sigma(z) = \frac{1}{1 + e^{-z}}$$

If we define $y' = X\beta$ and W to be diagonal matrix and the elements are from $y'(1 - y')$, the gradient $g(\beta)$ is $X^T(y - y')$ and Hessian $H(\beta) = -X^T W X$.

So β will be updated by $\beta' = \beta - H(\beta)^{-1} g(\beta)$.

Cox proportional hazard models. The Cox Proportional Hazards Model [214], often referred to as the Cox model, is a widely used statistical technique for analyzing time-to-event data. Here we give an overview of how Cox model works.

Suppose the input data is (X, T, E), where X consists of N patients and p covariates. T is a vector of N elements indicating the event time for each patient and E contains event indicator for each patient, e.g., whether that patient i is dead at timestamp T_i.

The Cox model assumes that the hazard ratio is proportional to a linear combination of covariates. That is

$$\frac{Hazard(X_i)}{Hazard(X_j)} = \frac{\exp(\beta^T X_i)}{\exp(\beta^T X_j)}$$

where $\beta = [\beta_1, \beta_2, ..., \beta_p]$ is a vector of length p, and β_i indicates the effect size of $i - th$

covariate on the hazard. So analysts are usually interested in what β is.

Another thing the analyst wants to learn about is how confident we are about the conclusion, which is captured by P value. In other words, the P value helps gauge whether the observed data could have occurred by random chance if there is no real effect.

In our target study, Newton-Raphson method is used to train the Cox model and Wald test is used to compute the P value. We will describe the process at a high level.

The partial log likelihood of Cox model is defined as

$$LL(\beta) = \log \Pi_{j \in D} \frac{\exp(\beta^T X_j)}{\sum_{l \in R_j} \exp(\beta^T X_l)}$$

where D contains the index of patients who are dead and R_j contains the index of patients who are at risk at timestamp T_j.

The goal is to find a β that maximize the likelihood and the method we use is Newton-Raphson. At a high level, if we define $H(\beta)$ to be the Hessian matrix and $g(\beta)$ to be the gradient, β will be updated by the following:

$$\beta' = \beta - H(\beta)^{-1} g(\beta)$$

If we expand $g(\beta)$,

$$g(\beta) = \sum_{j \in D} X_j - \sum_{j \in D} \frac{\sum_{l \in R_j} X_l \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)}$$

Similarly, if we define $a^{\otimes 2} = aa^T$ and we expand $H(\beta)$

$$H(\beta) = -\sum_{j \in D} \frac{\sum_{l \in R_j} X_l X_l^T \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)} + [\frac{\sum_{l \in R_j} X_l \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)}]^{\otimes 2}$$

With Newton-Raphson, we can get some β that maximize the log likelihood. The next

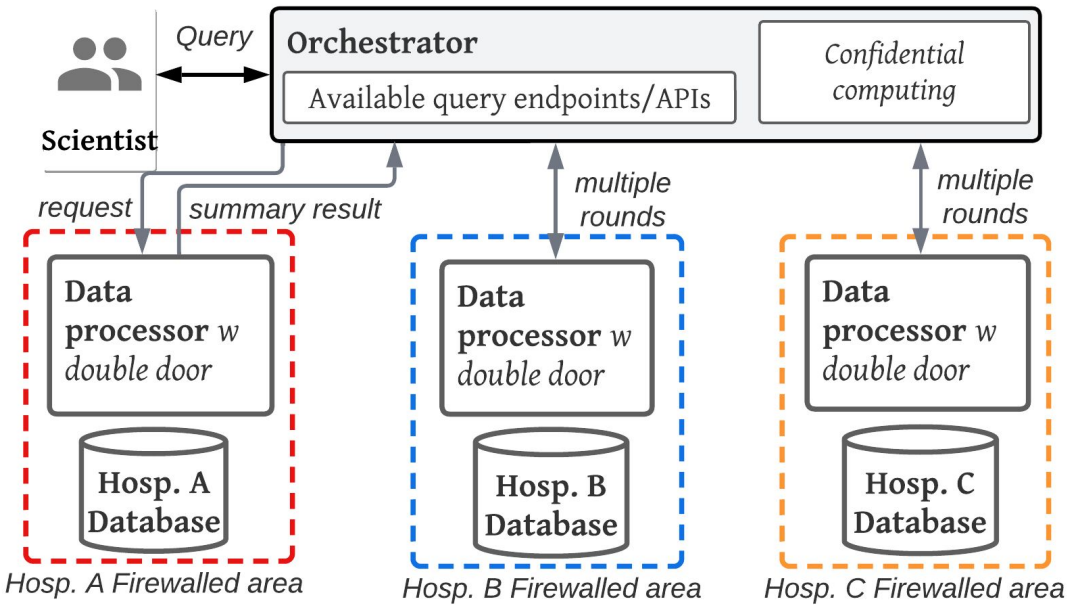


Figure 5.1: High-level architecture of the virtual pooling system.

step is to compute the P value.

The standard error of β, $se(\beta)$ is estimated to be diagonal elements of $H(\beta)^{-1}$ and P value can be derived from $\beta/se(\beta)$. That said, once we get Hessian and gradients of β, we can apply Newton-Raphson for training and Wald test to get the P value.

Descriptive statistics. Besides PSM and Cox model, virtual pooling also needs to support some descriptive statistics, including mean, variance, standard error and etc. These descriptive statistics help us normalize the data, analyze the possible issues with Cox training. For example, covariates with low variance can stop Cox model from converging and we might need to remove those covariates as a solution.

5.2 Overview of virtual pooling

5.2.1 Architecture of virtual pooling

The basic architecture of virtual pooling system has two parts: i) data processors; and ii) an orchestrator(Figure 5.1). Firewalled data processors run inside the firewalls maintained by hospital IT staff. Each processor connects to a hospital database and generates summary results according to the orchestrator's requests. Importantly, a data processor keeps individual-level data within the firewalls and uploads only summary results containing aggregated data.

Orchestrator runs on a third-party server. It receives a statistical query from a data scientist and executes it across the data processors in multiple rounds. In each round, it sends a sub-component of the query to each processor and receives summary results. The orchestrator combines the results at the end of a round, and begins the next round. This multi-round approach is crucial for the illusion of a pooled data repository.

5.2.2 Protocol overview

To begin an analysis pipeline, the analyst supplies input parameters (preprocessing function, covariates used for PSM and Cox model and parameters for matching algorithm) to the orchestrator.

In the preprocessing phase, the orchestrator sends the preprocessing function to health systems and each health system preprocess the data locally to get a local table for later computation. The orchestrator also coordinates between health systems to normalize the data for later logistic regression and Cox model training.

Next, in the PSM phase, the orchestrator trains a logistic regression model across all health systems securely and publish the model (β of logistic regression in §5.1.4). Each health system uses the β to compute the propensity scores (PS) and share the PS with the orchestrator. The

orchestrator combines all the PS and runs a matching algorithm locally based on all PS, and publish the matching results. That is, which indices of patients from which health systems are selected in the matched cohort. Each health system uses this information to form/select a local partition of matched cohort.

Finally, in the Cox model phase, the orchestrator trains a Cox PH model across all health systems securely and publish the model (β of Cox model in §5.1.4) as well as the P value.

5.3 Design of virtual pooling

At the core of virtual pooling is how descriptive statistics, PSM and Cox model are computed. We now go over the design details of virtual pooling piece-by-piece.

5.3.1 Preprocessing phase

In preprocessing phase, much of the computation happens locally, for example, filtering for patients who meet some conditions for analysis. The only part that requires coordination between multiple health systems is normalization and computation of descriptive statistics. Normalization is required for better convergence of logistic regression models and Cox PH models. Descriptive statistics, including mean and variance of each covariate, are useful for the analyst to decide whether some covariate should be included or not.

It's not difficult to compute mean and variance of each covariate in virtual pooling because we are allowed to reveal summary-level information. For example, to compute a global mean, we can ask each health system to share the sum of its local data as well as the number of rows. The server just needs to add up the sum from each health system and divide it by the total number of rows. There are descriptive statistics that cannot be computed this way, e.g., median. But since it's not used in our target analysis, virtual pooling doesn't support precise computation of it for now.

5.3.2 PSM phase

PSM consists of the calculation of propensity scores and a matching operation, and the goal of PSM is to balance the cohort between two groups, e.g., whether some medicine is used. There have been plenty of works on privacy-preserving logistic regression, based on techniques like differential privacy, secret sharing and homomorphic encryption. These methods don't work well for virtual pooling because we are allowed to share summary-level information, which makes the computation much easier. At a high level, virtual pooling runs a global Newton-Raphson training with only use of summary-level information and doesn't invovle cryptographic tools.

We give an overview of how virtual pooling trains a logistic regression model here. Recall that from a centralized view (§5.1.4), the gradient $g(\beta)$ is $X^T(y - y')$ and Hessian $H(\beta) = -X^T W X$. The β will be updated by $\beta' = \beta - H(\beta)^{-1} g(\beta)$. Since both gradients and Hessian are summary-level information, we can reveal them for the Newton-Raphson step. It's also easy to compute them using summary-level information. Assume health system i has $X_{[i]}, y_{[i]}, y'_{[i]}$,

$$g(\beta) = X^T(y - y') = \sum X_{[i]}^T (y_{[i]} - y'_{[i]})$$

That said, we just need to ask health system i share $X_i^T(y_i - y'_i)$ with the server, which is summary-level information and secure to share.

Similarly, if we follow the definition of W and define W_i to be diagonal matrix and the elements are from $y'_{[i]}(1 - y'_{[i]})$, it's not difficult to show that,

$$H(\beta) = -X^T W X = -\sum X_i^T W_{[i]} X_i$$

So each health system just needs to locally compute $X_{[i]}^T W_{[i]} X_{[i]}$ and share it with the server.

One training step will be as follows: i) the coordinator share β with all health systems; ii)

each health system computes $X_{[i]}^T(y_{[i]} - y'_{[i]})$, $X_{[i]}^T W_{[i]} X_{[i]}$ and share with the coordinator; iii) the coordinator combines to get $g(\beta)$ as well as $H(\beta)$, and update $\beta' = \beta - H(\beta)^{-1}g(\beta)$. The training repeats the above steps until convergence.

After we get a logistic regression model, we can compute propensity scores and do the matching. Each health system can compute $X_{[i]}\beta^T$ to get the scores and share the scores with the coordinator. Since the matching only takes as input the scores, it can happen locally at the coordinator. After the matching is done, the coordinator returns indices of which patients are selected to the health systems.

5.3.3 Cox model phase

We cannot train a Cox PH model like logistic regression where we only need to use summary-level information. If we follow a similar process like logistic regression, the short conclusion is that we can compute $g(\beta)$ using only summary-level information and also the first term of $H(\beta)$ (§5.1.4). However, the second term, $[\frac{\sum_{l \in R_j} X_l \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)}]^{\otimes 2}$ cannot be computed only by summary-level information.

If we formalize this problem a little bit, health system i has a matrix $A_i \in R^{d \times M}$, where M is the total number of deaths across all health systems and d is the number of features. Our goal is to compute the following without revealing any information about A_i since A_i contains patient-level information:

$$(\sum A_i)(\sum A_i)^T$$

We use shamir secret sharing [43] to compute this matrix multiplication securely because we want to tolerate compromising of a threshold of health systems. At a high level, each site i generates shamir secret sharing of A_i and sends the shares with other health systems. At the end of this step, we have $[A_i]$ in the secret shared form. We can compute $(\sum A_i)(\sum A_i)^T$

over the secret shared values and finally ask the health systems reconstruct the results with the coordiantor.

However, simply applying this protocol can be super expensive. For example, in our target study [2], there are 59 health systems and a total 38K deaths among 308K patients and 17 features, which means each $A_i \in R^{17\times 38K}$. Since we need to use 159-bit integer to hold floating point numbers for accuracy purpose, the sharing cost is estimated to be about 760 MiB per health system per Newton-Raphson step. The training usually takes about 5 steps, which means at least 3.8 GiB to train a Cox PH model for a health system. This cost is too high, compared with an average of 0.35 MiB of network transfer per health system in the plaintext centralized approach.

Tie optimization. This optimization is tailored for Cox model training. The key insight is each row of matrix A_i^T corresponds to some death event and can be divided into blocks of rows according to the death time, where each block can be generated by linear combination of two row vectors. Patients who have the same death time is called a tie. So instead of sharing A_i with other health systems, we can share row vectors for each block and ask health systems to expand to get $[A_i]$ in the shared form. The rest of computation is the same as before. With this optimization, the network cost reduces to about 3 MiB per Newton-Raphson step and about 15 MiB for the training, which is about $250\times$ improvement for our target dataset. For the detailed protocol description, we refer readers to the Appendix §C.1.

5.4 Implementation

Our goal is to re-run the analysis of a high-impact study [2] on the association between chronic immunosuppressive use and in-hospital outcomes of patients with COVID-19 [2] to verify that virtual pooling can support real-world statistical analysis with privacy and accuracy. However, the data cannot leave the enclave and we are limited to their single-server

Python environment, which makes it hard for us to build a distributed prototype. So in this section, we will present details of the analysis pipeline.

The original study analyzed the data from the National COVID Cohort Collaborative (N3C) repository, which contains EHR of patients in hospitals with confirmed or suspected COVID-19 in the USA. It has code publicly available[1], including data extraction code in Spark SQL and Python, analysis in SparkR.

We used their original data extraction code in the N3C Enclave and got a cohort of 334,754 patients from 59 health systems. This final analytic cohort is larger than the one used in the paper, which has 222,575 patients from 42 health systems. Due to the differences in the final analytic cohort, we did not compare virtual pooling's results to the study. Instead, we only compared the results of virtual pooling and direct pooling on the cohort of 334,754 patients, to verify virtual pooling's accuracy.

The original study used the R package matchIt [215] for propensity score matching. The propensity scores were computed using a logistic regression model. We followed the original study to take the use of immunosuppressive medicine as the regression outcome variable and the same set of covariates, including the week of admission, age, sex and etc, as regression covariates. The matching was done with a 4:1 nearest neighbor matching algorithm without replacement and a caliper of 0.2 pooled SDs of the estimated propensity score.

To estimate the risk of death in the matched cohort, the study used the R package survival [216] to train a Cox proportional hazards model. Similarly, we followed the study to use the same set of covariates, including immunosuppressive medicine use, congestive heart failure, renal disease, and etc.

[1] https://github.com/National-Clinical-Cohort-Collaborative/CS-ISC

5.5 Evaluation

In this section, we show how accurate virtual pooling is compared with centralized baseline.

Setup. To simulate the distributed setting, we split the final analytic cohort into 59 partitions, according to the covariate of the contributing data site. We used virtual pooling to run the same propensity score matching and Cox proportional hazards model training. Throughout the analysis, virtual pooling only transferred summary-level information across different data partitions.

5.5.1 End-to-end analysis results

To test the accuracy of virtual pooling against the pooled baseline, we compared the end-to-end results, namely the coefficients of covariates from the Cox PH model and also the P value of each covariate. We want to test whether virtual pooling can identify the same set of significant covariates and whether the coefficients of those covariates are identical to those from the baseline pooled analysis. Our analytic cohort has 334,754 patients from 59 health systems. For pooling, we ran propensity score matching and Cox PH model training. For virtual pooling, the cohort was distributed into 59 partitions according to the contributing health system and we ran propensity score matching and Cox PH model training on the distributed dataset. We compared the coefficients of the Cox PH model and P values of the covariates.

The pooled analysis has a log likelihood of -121302.60 and virtual pooling has the same log likelihood of -121302.60. As shown in Figure 5.2, virtual pooling has the same coefficients as pooled baseline and the P value are also the same.

PSM results We also compare the cohort formed by PSM. As discussed, PSM contains two steps: i) training a logistic regression model to get propensity scores and ii) nearest neighbor matching. Since we run the same matching algorithm based on the scores, so we just need to

	Baseline Coef	VP Coef	Baseline P value	VP P value
immuno_flag	-0.05	-0.05	1.83E-02	1.83E-02
transplant_any	-0.18	-0.18	<5.00E-07	<5.00E-07
pulmonary	0.11	0.11	<5.00E-07	<5.00E-07
renal	0.15	0.15	<5.00E-07	<5.00E-07
CHF	0.14	0.14	<5.00E-07	<5.00E-07
rheumatic	0.04	0.04	0.13	0.13
rx_other_pulm	0.04	0.04	0.04	0.04
rx_renal	-0.20	-0.20	<5.00E-07	<5.00E-07
rx_laba	-0.02	-0.02	0.52	0.52
dmcx	0.04	0.04	0.15	0.15
diabetes	-0.04	-0.04	0.14	0.14
rx_inhaled_cs	-0.09	-0.09	6.61E-06	7.00E-06
cancer	0.13	0.13	<5.00E-07	<5.00E-07
liver_mild	-0.04	-0.04	0.07	0.07
mets	0.06	0.06	0.05	0.05
PVD	0.06	0.06	0.00	0.00
rx_insulin	0.09	0.09	1.11E-05	1.10E-05

Figure 5.2: End-to-end results of centralized baseline and virtual pooling (VP).

compare the logistic regression.

The baseline uses the R package stats [217] for logistic regression, which uses iteratively reweighted least square method. Virtual pooling uses Newton-Raphson method, which is essentially the same as the iteratively reweighted least square for logistic regression.

For a precision of 1E-10, in virtual pooling, the logistic regression takes about 8 training epochs to converge. PSM takes in total 51 covariates for regression and we only present a subset of covariates in the Figure 5.3. As shown, the converged coefficients are identical between virtual pooling and pooling.

5.5.2 Robustness experiments

We used simulated datasets and compared virtual pooling with pooled baseline to evaluate whether virtual pooling has robustness accuracy. We sampled 8K rows from the N3C dataset and split them into three partitions to get different data distributions on each partition, and we ran propensity score matching and Cox PH model training using virtual pooling and direct

	Baseline Coef	VP Coef
(Intercept)	-3.4299979	-3.4299979
male	-0.0910278	-0.0910278
age	-0.1078902	-0.1078901
data_partner_id	0.0822743	0.0822743
CHF	-0.0174272	-0.0174271
PVD	-0.0009256	-0.0009255
pulmonary	0.3927734	0.3927734
rheumatic	0.3016752	0.3016752
liver_mild	0.0717490	0.0717490
diabetes	0.1273330	0.1273330
dmcx	-0.0388725	-0.0388725
renal	0.0760874	0.0760874
cancer	0.2640189	0.2640189
mets	0.1489797	0.1489797
rx_insulin	0.0810680	0.0810679
rx_metformin	-0.0149585	-0.0149585
rx_laba	0.0711858	0.0711857
rx_inhaled_cs	0.3319529	0.3319528
rx_other_pulm	0.1645172	0.1645172
rx_renal	0.1211015	0.1211015
transplant_any	0.5025364	0.5025364

Figure 5.3: Logistic regression coefficients of centralized baseline and virtual pooling (VP).

pooling.

For example, we picked a significant covariate rx_insulin and non-significant covariate immuno_flag in the pooled baseline and made three partitions have different correlation and distribution of these two covariates. We compared the P values of virtual pooling and direct pooling to test whether virtual pooling can still get the identical results as direct pooling. We also evaluated the P values that were computed by each local partition to evaluate whether virtual pooling is robust even when local partitions have different results from direct pooling.

To show that virtual pooling has robustness, we did some experiments as described in the method section. In short, we sample a small dataset with 8K patients and split them into 3 partitions (sites) to get different data distributions and run propensity score matching and Cox PH model. We presented two example experiments here.

First, we picked two covariates: rx_insulin and immuno_flag and we partitioned the data to make site 1 have 1K patients, site 2 have 2K patients and site 3 have 5K patients. If one

Covariate A = rx_insulin, Covariate B = immuno_flag	Correlation coefficient	Covariate A P-value	Covariate B P-value
Site 1	0.144216	Not Converged	Not Converged
Site 2	-0.113732	0.232816	0.037926
Site 3	0.127933	0.227707	0.426225
Baseline	0.137760	0.004775	0.134493
VP	0.137760	0.004775	0.134493
Covariate A = rx_insulin, Covariate B = duration	Cox model Coefficient	Covariate A P-value	-
Site 1	0.589279	0.232916	-
Site 2	-0.029733	0.941576	-
Site 3	0.531353	0.002690	-
Baseline	0.377524	0.004775	-
VP	0.377524	0.004775	-

Figure 5.4: Robustness experiments of virtual pooling (VP).

row has (rx_insulin, immuno_flag) to be either (0,0) or (1,1), it has a higher probability to be assigned to site 1 and otherwise higher probability to be assigned to site 2. The remaining rows are just assigned to site 3. As a result, these three sites have different correlation between these two covariates: site 1 has a correlation coefficient of 0.144216, site 2 of -0.029733 and site 3 of 0.531353.

As shown in Figure 5.4, if we train Cox PH model local on each site, site 1 failed to converge due to a low variance. Site 2 and Site 3 identified rx_insulin as non-significant covariate. Meanwhile pooled baseline and virtual pooling both identified rx_insulin as significant covariate. Similarly, Site 2 identified immuno_flag as significant covariate, which was identified as non-significant by pooled baseline and virtual pooling.

Further, we experimented with different distributions of the correlation between one covariate and the duration of the events. Similarly if one row has (rx_insulin = 1, duration < median duration) or (rx_insulin=0, duration > median duration), it has higher probability to be assigned to site 1 and otherwise site 2. The remaining rows are just assigned to site 3.

We compared local training on each site, pooled baseline and virtual pooling. All three sites gave quite different Cox coefficients for rx_insulin from pooled baseline, while virtual

pooling had the same result. Site 1 and Site 2 identified rx_insulin as a non-significant covariate, which was instead identified as significant by pooled baseline and virtual pooling (Figure 5.4).

5.5.3 Latency

As mentioned earlier, the data cannot leave the N3C enclave and we are also limited to N3C's environment, which makes it difficult for us to measure the CPU cost as well as latency. So we make a very rough estimation of end-to-end latency by adding up each component's running time.

In centralized baseline, PSM takes about 280 s, which is dominated by the matching operation. The logistic regression takes 6 iterations to converge and is fast. The following Cox PH model also takes less than 1 second. So in total the latency is about 4.73 min.

In VP, the federated logistic regression is estimated to take about 3 s. For a precision of 1E-10, VP takes 10 iterations. The computation time is about 2 seconds and if we assume each communication round takes about 30 ms, propensity score computation thus takes less than 3 seconds for virtual pooling. The matching takes about 280 s also since it's the same piece of code as centralized baseline. The Cox training in VP takes about 17 s with optimizations like Chinese remainder theorem (CRT) encoding and Barrett reduction [218].

5.6 Related work

To the best of our knowledge, we are not aware of any federated systems that can support PSM and Cox model yet. Even for Cox model training, virtual pooling is also the first system that can provide accuacy as centralized baseline while being secure.

A lot of related works either use approximation to avoid sharing patient-level information or propose a different function from vanilla algorithm. For example, ODACH [18] proposes

some surrogate likelihood function for estimation. However, since these works are different from vanilla definition, there are fundamental errors that we cannot get rid of. The accuracy guarantees (how different from centralized baseline) can be different for different cases. For example, ODACH failed to converge in some of our experiments where the centralized converged well. Due to the accuracy issue, we don't consider these works as closely related.

We have not seen many works about privacy-preserving vanilla Cox training. Some works fail to provide enough privacy guarantees. For example, WebDISCO [219] can achieve the same accuracy as direct pooling but can reveal patient-level information when some risk set at some timestamp contains only one patient. Though Li et al.(2023) [220] also uses some approximation, it claims to provide close enough results as centralized. However, it has a similar issue as WebDISCO that in some cases patient-level information will be leaked.

There are also some works using the cryptographic tools. For example, Yu et al. (2008) [221] projects higher dimensional features into lower dimension vectors for Cox training. However, this projection introduces errors and also it's difficult to compute P value as needed for the analysis. Some works use homomorphic encryption to train Cox models [222, 223] but these works have two issues: i) functions like exp need to be estimated and thus introduces errors; ii) none of them support calculation of P values.

5.7 Summary

Federated statistical analytics is getting significant attention in biomedical research because traditional centralized approach is difficult to operationalize due to privacy concerns and thus slowing the pace of clinical studies. One big challenge of current approximation methods, those that provide good privacy, is that they cannot give the same accuracy as the centralized baseline. Virtual pooling shows that one can perform statistical analytics with the same accuracy as centralized baseline without leaking patient-level information. Virtual

pooling improves the trade-off by focusing on two common methods, PSM and Cox, and tune the protocol to the threat model and these methods (§5.3). The main evaluation highlight is that virtual pooling has identical accuracy to centralized baseline and has robustness accuracy(§5.5).

Chapter 6

Conclusion and Future Work

Federated analytics has seen many applications in both industry and research communities since raw data can be kept locally and is friendly for privacy. However, federated analytics is not secure yet in the presence of adversaries and there is a tension between functionality, privacy and efficiency for privacy-preserving federated analytics systems. Existing works either sacrifice privacy for efficiency or ensure strong privacy guarantees with high or even impractical costs.

In this thesis, we take a step to answer the research question *Can we build federated systems that provide significant functionality, provide rigorous privacy guarantees, and are efficient at same time?* by focusing on three different types of real world queries. We take inspiration from existing works on privacy-preserving federated analytics systems and tune the system design and protocol to the target application to improve the trade-off between functionality, privacy and efficiency. Previous works with strong privacy guarantees [28, 30] have high and impractical costs for our target queries. The work in this thesis, designing and building federated analytics systems for our three target types of queries, shows that it's possible and practical to answer (a certain set of) federated analytics queries with strong privacy guarantees.

Our federated learning system, Aero, guarantees good accuracy, differential privacy over an untrusted server, and keeps the device overhead low. Aero tunes system architecture and design to a popular federated learning algorithms called FedAvg [48] and this tuning requires novel optimizations and techniques, e.g., a new protocol to securely aggregate updates from devices. Aero has comparable accuracy to plain federated learning (without differential privacy), and it improves efficiency (CPU and network) over state-of-the-art Orchard [28] by up to $10^5\times$.

Our federated graph analytics system, Colo, is a new, low-cost system that requires minutes of CPU time and a few MiBs in network transfers, for a particular subset of queries with strong privacy guarantees. At the heart of Colo is a new and tailored secure computation protocol that enables a device to securely and efficiently evaluate a graph query in its local neighborhood while hiding device data, edge data, and topology data. For running a variety of COVID-19 queries over a population of 1M devices, Colo's improvements of device-side costs over state-of-the-art Mycelium [30] are of up to three orders of magnitude.

Our federated statistical analytics system, virtual pooling, guarantees accuracy, privacy and keeps the latency low for multi-step statistical models (e.g. propensity score matching followed by causal inference). Virtual pooling tunes protocol to the statistical models like Cox PH models [214]. To the best of our knowledge, virtual pooling is the first federated analytics system that can support distributed training and inference of Cox PH model without sacrificing accuracy. Besides optimization for Cox PH models, virtual pooling is also a general-purpose framework that supports a set of different statistical methods with accuracy and strong privacy.

We envision numerous ways of expanding on research in privacy-preserving federated analytics and we lay out a number of potential directions as follows.

This thesis primarily focus on privacy and only guarantees correctness when all parties are honest-but-curious. Extending systems to efficiently guarantee correctness in the pres-

ence of malicious adversary can substantially improve usability. If we take federated learning as an example, ensuring correctness can prevent from poisoning attacks to the global model. One possible approach to provide correctness in the federated learning setting is robust federated learning, e.g., using median instead of mean to update the global model. The difficulties would be the efficiency since homomorphic encryption schemes are expensive for comparison operationss. So possible directions can be designing crypto-friendly robust federated learning algorithms or tuning the system and protocols to some robust FL method, e.g., using comparison-friendly encryption schemes. As for graph queries and statistical queries, we are not aware of robustness aggregation yet but this can also be interesting future work.

The span of queries we can support is still fairly small when considering the space of all possible analytics techniques that can be deployed at scale. Extending systems to support more queries can have huge impact. For example, Aero only supports DP-FedAvg and there have been many more federated learning algorithms, like DP-FTRL [33]. Also, Colo only supports a small set of graph queries and are limited to the input domain size. Possible directions can be extending to queries with larger input domain size. Also, in virtual pooling, future work can be to support more complicated statistical models privately and efficiently. Future work can also be to support more types of queries. For example, graph neural network (GNN) is gaining attention and extending Aero or Colo to support GNN training is interesting.

Though our systems all assume strong and practical threat model, there is still room to improve. For example, Aero assumes a small fraction (e.g., 3%) of devices can be compromised, and future work can be done to improve this fraction to a much larger threshold like 20%. Also in Colo, we assume a set of servers and at most a fraction (e.g., 20%) of them can collude with each other. Extending Colo to a single-server architecture will also be interesting. In virtual pooling, we assume at most 29 of 59 health systems can be compromised and these health systems are honest-but-curious. Building systems that are efficient and private against malicious health systems can also be one direction.

When building systems, we use cryptographic tools as building blocks and they are usually the bottlenecks. One direction can be to tailor these building blocks and implementation for better performance. For example, in Aero, device-side cost is dominated by the ZKP of HE ciphertexts. One possible direction can be tailoring the ZKP for this special circuit, e.g., using hardware to accelerate or optimizing ZKP schemes.

Though Aero and Colo are not general-purpose, they can support a type of queries. Instead, virtual pooling is designed for some specific algorithms, e.g., PSM and Cox. This limitation comes from the functionality requirement. For example, we can apply algorithms like SGD to train Cox PH models. Though there might be some accuracy loss compared to the centralized baseline, we don't need to propose and design a protocol for it. However, besides training the model, we also need to do the inference, which requires us to compute the Hessian matrix. There is no general-purpose way to compute Hessian matrix for different statistical models. So one direction can be extending virtual pooling to be general-purpose. This can be some methods to estimate P value in a more general way or efficient general-purpose computation, e.g., efficient HE.

Besides directions in terms of improving functionality, privacy and efficiency, there are also deployment problems. For example, Aero and Colo, as prototypes, don't consider training latency, resilience to drop-outs, stragglers, and etc. These problems are not trivial. For example, Aero, requires synchronization between billions of devices to form committees and millions of devices to build summation tree. Though Aero reuses committees, the end-to-end training time can still be a lot, which hurts usability of Aero. Similarly, Colo doesn't support interactive queries due to latency as well. Colo is not resilient to drop-outs either, which is quite common for mobile devices. Now Colo is designed for queries that are run infrequently and thus devices just need to be online, e.g., every day. However, analysts would prefer interactive queries for better efficiency of analysis. Improving latency will also make the resilience to drop-outs a huge problem.

Appendix A

Aero Supplementary

A.1 Privacy proof

The goal of Aero is to provide differential privacy for a class of federated learning algorithms. We will take DP-FedAvg as an example and prove that Aero indeed meets its goal in multiple steps. The proof for other algorithms such as DP-FedSGD is similar. The outline of the proof is as follows.

First, we will introduce a slightly modified version of DP-FedAvg that makes explicit the behavior of the malicious aggregator and devices. This modified version changes line 8 and line 12 in Figure 3.2. For instance, we will modify line 8 in Figure 3.2 to show that a byzantine aggregator may allow malicious devices to be sampled in a round. Appendix A.1.1 introduces the modified version and shows that the changes do not impact DP-FedAvg's differential privacy guarantee.

Next, we will prove that Aero executes the modified DP-FedAvg algorithm faithfully, by showing that enough Gaussian noise will be added (Appendix A.1.2) and if an adversary introduces an error into the aggregation, it will be caught with high probability (Appendix A.1.3).

Finally, we will prove that after aggregation, Aero's decryption protocol does not leak any information beyond the allowed output of DP-FedAvg (Appendix A.1.4).

We will not cover security of the protocols used in the setup phase, e.g., the sortition protocol to form committees, because Aero does not innovate on these protocols. With the above proof structure, we will show Aero provides the differential privacy guarantee to honest devices' data.

Before proceeding to the proof, we introduce a few definitions. In the main body of the paper, for simplicity, we did not distinguish between honest-but-offline and malicious devices for the DP-noise committee (§3.3.2). Instead, we considered all offline devices as malicious for this committee, since it's not possible to tell whether an offline device is malicious or not.

However, for devices that generate updates, we do protect honest-but-offline devices' data. So when talking about generator devices, we use the following terms:

- honest-and-online devices,

- honest-but-offline devices,
- honest devices: including both honest-and-online and honest-but-offline devices
- malicious devices

As for the DP-noise committee members, we still follow the previous definition, namely

- honest members: honest and online members
- malicious members: malicious or honest-but-offline members

A.1.1 DP-FedAvg

As mentioned above, Aero must take into account the behavior of malicious entities for the computation in line 8 and line 12 of Figure 3.2.

There are two reasons, Aero must modify line 8: first, a malicious aggregator may filter out honest-but-offline devices' data in the aggregation; second, a malicious aggregator may add malicious devices' data in the aggregation.

To capture the power of a malicious aggregator, we change line 8 to be

$C^t \leftarrow$ *subset of (sampled honest users with probability q) + (some malicious devices)*

We must modify line 12 because the DP-noise committee is likely to add more noise as noted in the design of the generate phase (§3.3.2). To capture this additional noise, we change line 12 to be

$\Delta^t \leftarrow \sum_k \Delta_k^t + \mathcal{N}(0, I\sigma^2)$ *+ some additional bounded noise*

In the remainder of this section, we will prove that the modifications preserve the privacy guarantee of the original DP-FedAvg algorithm [224].

Device sampling (line 8). For device sampling, there are two possible attacks: either some malicious devices' data will be included or some honest devices' data will be filtered out.

The first case is not a problem, since it's equivalent to post-processing: for example, after aggregation, the malicious aggregator can add malicious updates. Post-processing of a deferentially private result does not affect the differential privacy guarantee (this follows from the post-processing lemma in the differential privacy literature [225, 42]).

To prove that filtering out honest devices' data will not impact the privacy guarantee, we'll first give intuition and then a rigorous proof. Intuitively, a larger sampling probability (larger q) leads to more privacy loss, because a device's data is more likely to be used in training. So informally, if each device is expected to contribute an update fewer times, the privacy loss is expected to be less. Now coming back to the case where the aggregator filters out honest devices' data, it is obvious that each device is expected to contribute updates no more frequently than without filtering, which means the privacy loss is expected to be no more than without filtering.

In the original paper of DP-FedAvg [224], the DP guarantee relies on the moments accountant introduced by Abadi et al. [55], whose tail bound can be converted to (ϵ, δ)-differential privacy. To be precise, the proof uses a lemma (which we will introduce shortly) that gives

the moments bound on Gaussian noise with random sampling, which is equivalent to (ϵ, δ)-differential privacy. We cite this lemma as Lemma A.1.1 in the appendix.

So next we will show by replacing the original random sampling with random sampling plus filtering, the moment bound still holds. In the discussion, without losing generality, we will focus on functions whose sensitivity is 1, for example, the UserUpdate function that does local training and gradient clipping with $S = 1$ in DP-FedAvg (Figure 3.2). Notice that for any function f' whose sensitivity is $S' \neq 1$, we can always construct a function $f = f'/S'$ whose sensitivity is 1.

Lemma A.1.1 *Given any function f, whose norm $\|f(\cdot)\|_2 \leq 1$, let $z \geq 1$ be some noise scale and $\sigma = z \cdot \|f(\cdot)\|_2$, let $d = \{d_1, ..., d_n\}$ be a database, let $\mathcal{J}$ be a sample from $[n]$ where each $i \in [n]$ is chosen independently with probability $q \leq \frac{1}{16\sigma}$, then for any positive integer $\lambda \leq \sigma^2 \ln \frac{1}{q\sigma}$, the function $\mathcal{G}(d) = \sum_{i \in \mathcal{J}} f(d_i) + \mathcal{N}(0, \sigma^2 \boldsymbol{I})$ satisfies*

$$\alpha_{\mathcal{G}}(\lambda) \leq \frac{q^2 \lambda(\lambda+1)}{(1-q)\sigma^2} + O(q^3\lambda^2/\sigma^3).$$

Notice that if the bound on $\alpha_{\mathcal{G}}$ does not change when some data points are filtered out after sampling, the differential privacy guarantee will not change either. We refer readers to [55] for more details.

Claim A.1.1 *Let $\mathcal{J}$ be a sample from $[n]$ where each $i \in [n]$ is chosen independently with probability $q \leq \frac{1}{16\sigma}$. Let $\mathcal{J}' \subseteq \mathcal{J}$ be some arbitrary subset of $\mathcal{J}$. Then for any positive integer $\lambda \leq \sigma^2 \ln \frac{1}{q\sigma}$, the function $\mathcal{G}'(d) = \sum_{i \in \mathcal{J}'} f(d_i) + \mathcal{N}(0, \sigma^2 \boldsymbol{I})$ also satisfies*

$$\alpha_{\mathcal{G}'}(\lambda) \leq \frac{q^2 \lambda(\lambda+1)}{(1-q)\sigma^2} + O(q^3\lambda^2/\sigma^3).$$

Proof: Let's abuse the notation a little bit and define $\mathcal{G}_{f,\mathcal{J}}$ to be

$$\mathcal{G}_{f,\mathcal{J}}(d) = \sum_{i \in \mathcal{J}} f(d_i) + \mathcal{N}(0, \sigma^2 \boldsymbol{I}).$$

To prove this claim, since by definition $\mathcal{G}'(d) = \mathcal{G}_{f,\mathcal{J}'}(d)$, we just need to show

$$\alpha_{\mathcal{G}_{f,\mathcal{J}'}}(\lambda) \leq \frac{q^2 \lambda(\lambda+1)}{(1-q)\sigma^2} + O(q^3\lambda^2/\sigma^3).$$

To do this, suppose for now we have a f' on $\mathcal{J}$ whose sensitivity is no greater than 1 and gives the same output as f on $\mathcal{J}'$, namely

$$\|f'(\cdot)\|_2 \leq 1, \mathcal{G}_{f',\mathcal{J}}(d) = \mathcal{G}_{f,\mathcal{J}'}(d).$$

By applying Lemma A.1.1 on $\mathcal{G}_{f',\mathcal{J}}$, we get

$$\alpha_{\mathcal{G}_{f',\mathcal{J}}}(\lambda) \leq \frac{q^2\lambda(\lambda+1)}{(1-q)\sigma^2} + O(q^3\lambda^2/\sigma^3).$$

Since $\mathcal{G}_{f',\mathcal{J}}(d) = \mathcal{G}_{f,\mathcal{J}'}(d)$,

$$\alpha_{\mathcal{G}_{f',\mathcal{J}}}(\lambda) = \alpha_{\mathcal{G}_{f,\mathcal{J}'}}(\lambda).$$

Combining the preceding two equations, we get

$$\alpha_{\mathcal{G}_{f,\mathcal{J}'}}(\lambda) = \alpha_{\mathcal{G}_{f',\mathcal{J}}}(\lambda) \leq \frac{q^2\lambda(\lambda+1)}{(1-q)\sigma^2} + O(q^3\lambda^2/\sigma^3).$$

The remaining task is to construct such a f' on $\mathcal{J}$, where $\|f'(\cdot)\|_2 \leq 1$, to give the same output as f on $\mathcal{J}'$. One possible f' is as follows:

$$f'(d_i) = \begin{cases} f(d_i) & \text{if } i \in \mathcal{J}', \\ 0 & \text{if } i \in \mathcal{J} - \mathcal{J}'. \end{cases}$$

It's easy to prove $\sum_{i\in\mathcal{J}'} f(d_i) = \sum_{i\in\mathcal{J}} f'(d_i)$.

Next, for the sensitivity of f', it is not difficult to see $\|f'(\cdot)\|_2 \leq \|f(\cdot)\|_2$, since by removing or adding one entry to the database, f' will incur either the same change as f or no change. ■

So far we have proved that if only a subset of selected devices are included in aggregation or more malicious devices' data is included, the differential privacy guarantee will not be impacted.

Gaussian Noise (line 12). Recall that we also add some additional noise in line 12 in Figure 3.2. For the Gaussian noise, we need to prove additional noise will not impact privacy, which is not difficult to show, since this change is also equivalent to post-processing.

With the above proof, we've showed that our modified DP-FedAvg provides the same privacy guarantee as the original DP-FedAvg.

A.1.2 DP-noise committee

This section shows that the $\mathcal{N}(0, I\sigma^2)$ part of line 12 of our modified DP-FedAvg is executed faithfully.

We have already covered in the generate phase (§3.3.2) that $\mathcal{N}(0, I\sigma^2)$ amount of noise will be generated by the DP-noise committee as long as the DP-noise committee does not violate its threshold: less than A out of C devices of the DP-noise committee are indeed malicious. Thus, in this section we will derive the probability of a committee having fewer than some threshold of honest members. We will follow the same way to compute the probability as in Honeycrisp [29], which is a building block for Orchard [28].

Claim A.1.2 *(Aero) If a randomly sampled DP-noise committee size is C, the probability of a*

committee member being malicious is f, the probability that committee has fewer than $(1-t)\cdot C$ honest members is upper-bounded by $p = e^{-fC}(\frac{ef}{t})^{tC}$, when $1 > t \geq f$.

Proof: We treat each member being malicious as independent events. Let X_i be a random variable

$$X_i = \begin{cases} 1, & \text{if member i is malicious,} \\ 0, & \text{if member i is honest.} \end{cases}$$

Let $X = \sum_i X_i$ be the random variable representing the number of malicious members. If $t \geq f$, the Chernoff bound shows that

$$Pr(X \geq tC) \leq e^{-fC}(\frac{ef}{t})^{tC}.$$

So the probability of fewer than $(1-t)\cdot C$ members being honest will be upper-bounded by p. ■

With (high) probability, the lower bound of honest committee members will be $(1-t)C$. As long as each honest member contributes $\frac{1}{(1-t)C}\cdot\mathcal{N}(0,\sigma^2\boldsymbol{I})$ noise, the total amount of noise will be no less than $\mathcal{N}(0,\sigma^2\boldsymbol{I})$.

To give some examples of what committee sizes could be, when $f = 0.03$, we may set $t = 1/7, C = 280$, to achieve $p = 4.1\cdot 10^{-14}$; when $f = 0.05$, we may set $t = 1/8, C = 350$ to achieve $p = 9.78\cdot 10^{-7}$ or $C = 450$ to achieve $p = 1.87\cdot 10^{-8}$; and, when $f = 0.10$, we may set $t = 1/5, C = 350$ to achieve $p = 1.34\cdot 10^{-6}$ or $C = 450$ to achieve $p = 2.82\cdot 10^{-8}$.

A.1.3 Aggregation

This section will prove that the additions in line 12 in our modified DP-FedAvg are executed faithfully in Aero. Otherwise, the aggregator will be caught with a high probability.

We will first prove the integrity of additions as in Aero's add phase protocol and in Honeycrisp. Next, we will prove the freshness guarantee that no ciphertexts from previous rounds can be included in the current aggregation. Finally, we will prove how these two proofs together show that line 12 of modified DP-FedAvg is executed faithfully by Aero.

Integrity. We will start with the integrity claim from Honeycrisp [29].

Claim A.1.3 *At the end of add phase, if no device has found malicious activity by the aggregator $\mathcal{A}$, the sum of the ciphertexts published by $\mathcal{A}$ is correct (with high probability) and no inputs of malicious nodes are dependent on inputs of honest nodes (in the same round).*

Proof: We refer readers to Honeycrisp [29] for more details. Here we will just give a short version for demonstration.

We will first prove no inputs of malicious devices are dependent on inputs of honest devices in the same round. Next we will prove if a malicious aggregator introduces an error into a summation tree, it will be caught with high probability.

First, assume for the sake of contradiction that it is possible for a malicious device to set its ciphertext to be c that is from an honest device in the current round. The adversary needs to produce a $t = Hash(r||c||\pi)$ and include t in the Merkle tree MC, before the honest device reveals c. Under the Random Oracle assumption in cryptography, this is not possible. So no inputs of malicious devices are dependent on inputs of honest devices in the same round.

Next we need to show if $\mathcal{A}$ introduces an error into a summation tree ST, it will be caught with high probability. In particular, here we will just show the case where $\mathcal{A}$ introduces an error into one of the leaf nodes. Similar analysis can be done for non-leaf nodes, which is presented in Honeycrisp [29].

Suppose the total number of verifiers is W and the total number of leaf nodes in one summation tree is M' (Figure 3.4). Here we make the assumption that $M' \approx qW$, where q is the sampling probability. We will prove this assumption later in this section.

Suppose $\mathcal{A}$ introduces an error into a particular leaf node $j \in [0, M'-1]$. The probability of an honest device picking any v_{init} to have $j \in [v_{init}, v_{init}+s]$ is

$$\frac{qs}{M'}.$$

Since there are at least $(1-f)W$ honest devices, the probability of no honest device checking j is

$$(1-\frac{qs}{M'})^{(1-f)W} = (1-\frac{s}{W})^{(1-f)W} \leq e^{-(1-f)s}.$$

Similarly we can prove if $\mathcal{A}$ introduces error into non-leaf nodes, it will be caught with a high probability. For instance, if $f = 3\%, s = 5$, the probability is about 0.007; if $f = 3\%, s = 20$, the probability is about 10^{-11}.

■

As noted above, proof relies on an assumption about the threshold of Sybils (pseudonym leaf nodes) the aggregator can introduce into the summation tree. Intuitively, if the aggregator can introduce as many Sybils as it wants to make $M' \gg qW$, then the probability of each node being covered will decrease by a large factor, thus impacting privacy. For example, if $W = 100, q = 0.01$, M' is expected to be close to $qW = 1$ and the 100 verifiers expect to verify only 1 leaf node. However, if the aggregator introduces 99 additional leaf nodes into the summation tree while the 100 verifiers still expect to verify only 1 leaf node, obviously many malicious nodes are very likely to be missed by the verifiers. Before claiming in Aero there is a threshold of Sybils, we need to introduce an assumption from Honeycrisp [29], which we will also use in our claim.

Assumption A.1.1 *(Honeycrisp) All devices know an upper bound W_{max} and a lower bound W_{min} of the number of potential participating devices in the system. If the true number of devices is W_{tot}, then by definition: $W_{min} \leq W_{tot} \leq W_{max}$. We assume $\frac{W_{max}-W_{tot}}{W_{min}}$ is always below some constant (low) threshold (this determines the portion of Sybils the aggregator $\mathcal{A}$ could make without getting caught). $\frac{W_{max}}{W_{min}}$ should also be below some (more generous) constant threshold.*

With the above assumption, similarly, we claim in Aero:

Claim A.1.4 *(Aero) There is a upper bound on the portion of Sybils a malicious aggregator can introduce without being caught. Precisely, all devices know an upper bound M_{max} and a lower bound M_{min} of the number of potential leaf nodes in the summation tree. If the true number of leaf nodes is M_{tot}, then by definition: $M_{min} \leq M_{tot} \leq M_{max}$. In Aero, $\frac{M_{max}-M_{tot}}{M_{min}}$ is always below some constant (low) threshold (this determines the portion of Sybils the aggregator $\mathcal{A}$ could make without getting caught). $\frac{M_{max}}{M_{min}}$ will also be below some (more generous) constant threshold.*

Proof: Note that in this claim, by "leaf nodes", we mean leaf nodes corresponding to a honest generator device. Once we know a upper bound M_{max} and a lower bound M_{min} of the number of leaf nodes, in the worst case, a malicious aggregator could introduce at most $M_{max} - M_{min}$ Sybils. If the true number of leaf nodes is M_{tot}, then the aggregator could introduce at most $M_{max} - M_{tot}$ Sybils.

Let's first consider M_{min} and M_{max}.

Informally, if the number of devices W is large enough, the number of leaf nodes (generators) is likely to be close to the expectation qW. Suppose for now there are two constants capturing this "closeness": k_{min}, k_{max}, where $k_{min} \approx 1, k_{max} \approx 1$. Then, we have

$$k_{min} \cdot qW_{min} \leq M_{min} \leq k_{max} \cdot qW_{min},$$

$$k_{min} \cdot qW_{max} \leq M_{max} \leq k_{max} \cdot qW_{max}.$$

Consider the fraction $\frac{M_{max}}{M_{min}}$.

$$\frac{M_{max}}{M_{min}} \leq \frac{k_{max} \cdot qW_{max}}{k_{min} \cdot qW_{min}} = \frac{k_{max}}{k_{min}} \cdot \frac{W_{max}}{W_{min}}$$

$\frac{W_{max}}{W_{min}}$ is below some constant threshold as in Assumption A.1.1. If $\frac{k_{max}}{k_{min}}$ is smaller than some constant threshold, then it's reasonable to assume like in Honeycrisp that $\frac{M_{max}}{M_{min}}$ is below some constant threshold. We will discuss the values of k_{max} and k_{min} at the end of the proof.

As for the fraction $\frac{M_{max}-M_{tot}}{M_{min}}$, it's not difficult to see that,

$$\frac{M_{max} - M_{tot}}{M_{min}} \leq \frac{k_{max}W_{max} - k_{min}W_{tot}}{k_{min}W_{min}}$$

Similarly if both k_{max} and k_{min} are close to 1, this fraction will be close to $\frac{W_{max}-W_{tot}}{W_{min}}$, which is below some small constant (low) threshold as specified in Assumption A.1.1. So in Aero, it is reasonable to assume this fraction is also below some constant (low) threshold.

Lastly, let's discuss k_{max} and k_{min}. Consider a general case where the total number of devices is W and the sampling probability is q. The number of sampled devices (leaf nodes) is X. Chernoff bound states that,

$$Pr(X < k_{min}qW) < (\frac{e^{k_{min}-1}}{{k_{min}}^{k_{min}}})^{qW}.$$

With $qW = 5000$, $k_{min} = 0.9$, the above probability will be smaller than $5.77 \cdot 10^{-12}$.

Similarly, with $qW = 10000, k_{min} = 0.93$, the probability will be smaller than $1.27 \cdot 10^{-11}$. Since Aero is designed for large-scale training, it's reasonable to assume k_{min} is close to 1. The argument is similar for k_{max} (e.g. $k_{max} = 1.01$).

However, if qW is, for example, 500, to achieve a similar probability of $1.17 \cdot 10^{-11}$, k_{min} will be about 0.7. In this setting, it's not reasonable to assume $M' = qW$ as in Claim A.1.3 any more. Instead one will need to assume a different bound, e.g. $M' < 2qW$, and re-calculate the probability. For example, if $M' = 2qW$, in Claim A.1.3, the probability of an honest device checking a particular leaf node will still be the same. However, the probability of no honest device checking a particular node will be instead

$$(1 - \frac{qs}{M'})^{(1-f)W} < (1 - \frac{s}{2W})^{(1-f)W} \leq e^{-(1-f)s/2}$$

Verifiers are expected to verify more nodes in the summation tree to make Claim A.1.3 true and thus ensure privacy. ■

Freshness. Reusing the encryption key will not affect the security proof in each round, but will lead to attacks across rounds, where information from previous rounds can be leaked. The adversary may obtain the victim device's ciphertext $Enc(m)$ from a previous round and use $k \cdot Enc(m)$, where k is a large constant, to participate in a later round. Since k is large, $k \cdot Enc(m)$ will dominate the aggregated result. After decryption, the adversary will be able to learn approximately m. We define the freshness as the guarantee that only fresh generated ciphertexts are included in the aggregation.

We need to prove that by asking each device to put the round number in the first slot of ciphertext, generating corresponding ZK-proof, and in the add phase verifying the ZK proof, the adversary will not be able to use ciphertexts from previous rounds in the current round.

First, according to the knowledge soundness property of zkSNARK, which states that it is not possible for a prover to construct a proof without knowing the witness (e.g. secret inputs), an adversary can't construct a proof for a new round [226]. This means the adversary can only insert a non-valid proof into the summation tree.

Next, if the adversary inserts one ciphertext from a previous round with a non-valid proof into the summation tree, with high probability, it will be caught, since as claimed before, with high probability each leaf node will be checked by some honest devices. The honest devices will be able to detect this error.

How Aero supports modified DP-FedAvg. We've covered the integrity and freshness of Aero's add phase/aggregation. Next we'll show the modified DP-FedAvg will be executed faithfully. In order to show this, we claim the following:

Claim A.1.5 *Data from honest generator devices will be included at most once in the aggregation; data from honest DP-noise committee members will be included exactly once.*

Proof: It's not difficult to see that both honest generators' and honest DP-noise committee members' data will be included at most once, as defined in Claim A.1.3. So for honest generator devices, the proof is done.

As for honest DP-noise committee member, we just need to prove data from an honest member will be included (at least once). Recall that for a honest committee member, it is always online during a round, otherwise will be considered as malicious. Also recall that in the add phase, after the aggregator constructs the summation tree and the Merkle tree, the aggregator needs to send a Merkle proof to the device that its data is included in the summation tree (Step 7 in Figure 3.4). An honest committee member can thus make sure its data is included in the aggregation by verifying this Merkle proof. ■

This claim captures the requirement for faithfully executing the modified DP-FedAvg.

Firstly, data from honest generators (including online and offline) will be included at most once, which is exactly what our new sampling method does (Appendix §A.1.1): some honest generators might be filtered while others not; those included in the aggregation will be added exactly once.

Secondly, data from honest DP-noise committee members (only online) will be included exactly once, which ensures that enough Gaussian noise will be added.

Combining with the integrity and freshness of the underlying aggregation, the modified DP-FedAvg protocol will be executed as it is in Aero.

A.1.4 Decryption

The last step in Aero's protocol is decryption. In this section, we'll prove Aero's decryption protocol will not leak any information, except the decrypted result.

Let's first review the BFV scheme [46, 45]. Let the field for the coefficients of the ciphertext polynomials be Q, the polynomials themselves be from a polynomial ring R_Q, the distribution ϕ for the coefficients of error polynomials be the required Gaussian distribution ϕ (standard deviation=3.2), and the secret key sk be a polynomial of same degree N as the ciphertext polynomials but with coefficients from the ternary distribution ({-1,0,1}), which we denote as ψ. Then, given a small constant $\gamma \ll 1$, the BFV scheme has the following procedures

- $Keygen$: $s \leftarrow \psi, a \leftarrow R_Q, e \leftarrow \phi$. Compute $b = as + e$ and output $pk = (a, b)$ and $sk = s$
- $Enc(pk, m)$: $e_1 \leftarrow \phi, e_2 \leftarrow \phi, r \leftarrow \psi$, output $(c_1 = ar + e_1, c_2 = br + e_2 + m/\gamma)$
- $Dec(c_1, c_2)$: output $m = \lfloor (c_2 - c_1 s) \cdot \gamma \rceil$

Since m will finally be known to the aggregator and revealing m is safe, without losing generality, assume (c_1, c_2) is the encryption of 0. Further, as defined in release phase (§3.3.4),

$$e_{small} = c_2 - c_1 \cdot s = re + e_2 - se_1.$$

This small error must remain hidden during the decryption process; otherwise, it may reveal information about the secret key s or the polynomial r. To hide e_{small}, Aero's scheme applies the smudging lemma [81]. This lemma states that to achieve $2^{-\lambda}$ statistical distance between e_{small} and $e_{small} + e_{smudging}$, $e_{smudging}$ just needs to be sampled from a uniform distribution

whose bound is λ bits more than the upper bound of e_{small}. Suppose the smudging distribution is ϕ', which is some uniform distribution whose bound is λ bits more than the upper bound of e_{small}.

Recall that in the release phase, the decryption committee reveals $c_1 \cdot s + e_{smudging}$ to the aggregator, where $e_{smudging} \leftarrow \phi'$. So the adversary's view is

$$(c_2, c_1, c_1 s + e_{smudging}),$$

which is equivalent to

$$(c_2, c_1, -c_2 + c_1 s + e_{smudging}).$$

If we can prove this view is indistinguishable from

$$(c_2, c_1, e'_{smudging})$$

where $e'_{smudging}$ is some freshly sampled error from the smudging distribution ϕ', then revealing $c_1 \cdot s + e_{smudging}$ to the aggregator will not leak more information than telling the aggregator a uniformly random number, since (c_1, c_2) are already known to the aggregator.

To prove the above claim, let's start with

$$(c_2, c_1, -c_2 + c_1 s + e_{smudging}).$$

Expanding c_2 and c_1, we get that the above is same as

$$(asr + re + e_2, ar + e_1, -re - e_2 + se_1 + e_{smudging}).$$

With the smudging lemma, the above is indistinguishable from

$$((as + e)r + e_2, ar + e_1, e'_{smudging}).$$

Notice that $e'_{smudging}$ has nothing to do with the secret key s, and we can apply Ring-LWE assumption to convert $(as+e, a)$ back to (b, a) as otherwise $(as+e, a)$ is not indistinguishable from (b, a). The above is indistinguishable from

$$(br + e_2, ar + e_1, e'_{smudging}) = (c_2, c_1, e'_{smudging}).$$

Since $e'_{smudging}$ doesn't depend on either the secret key or honest devices' data, revealing partial decryption result will not leak information about either the secret key or honest devices' data.

A.1.5 Details of the setup phase

During the setup phase, Aero (i) forms the master committee, which then (ii) receives and validates inputs for the round, and (iii) generates keys for cryptographic primitives. We present the details for only the second piece here, as the first and the third are discussed in

detail earlier (§3.3).

For the second piece, the master committee needs to check whether there is enough privacy budget to run a training task before launching it. To do this, the committee members need to calculate the new DP parameters before they launch the training task and check whether the new parameters are below some threshold. The details are as follows.

Recall that once the master committee is formed, each committee member receives the model parameters θ^t for the current round t, the user selection probability q, noise scale z, and clipping bound S from the aggregator for this round of training (required by DP-FedAvg; Figure 3.2).

Each committee member locally computes new values of the DP parameters ϵ, δ using the moment accounting algorithm $\mathcal{M}$ (line 14 in DP-FedAvg). This computation requires the DP parameters ϵ', δ' from round $t-1$ in addition to the inputs z, q. The committee member downloads the former from a public bulletin board, where it is signed by more than a threshold of honest members of the previous round's master committee. After getting the former DP parameters, the committee member calculates the new ϵ, δ. If the new values are below their recommended value, the committee member signs a certificate containing the parameters (θ^t, q, z, S), new values of ϵ, δ, and keys for cryptographic primitives and publish it to the bulletin board to start this training task.

Appendix B

Colo Supplementary

B.1 Privacy analysis

In this section, we show the privacy analysis of Colo. We sketch the proof as follows. We will first prove that if the local aggregation does not leak any information about node data, edge data, or the topology data and if the local aggregation evaluates a function F', which has some same property as the function F given by the query, the three phases of query distribution, local aggregation, and global aggregation will not leak information about the devices more than the query output. Later, we will prove that the local aggregation protocol meets the above requirement.

Before going into the proof, we list the assumptions we've made for privacy.

Assumption B.1.1 *The queries themselves, including the transformation and the metadata (the bounds on inputs and outputs), won't break the devices' data privacy.*

Since the queries are public, we assume we can meet this requirement by asking experts to review and certify the queries before execution, or only support a list of safe queries, as well as a list of safe transformation functions.

Assumption B.1.2 *We assume that devices can send messages to each other privately using a metadata private messaging system.*

In Colo, we use Karaoke [148] to meet this assumption. Other systems that can provide this functionality will also work and we are not limited to Karaoke.

Assumption B.1.3 *We assume that the adversary is not able to compromise all servers.*

Indeed, to use Karaoke [148], Colo assumes that 80% of the M servers (M is 40-100) remain honest. (Only with this assumption Karaoke provides its guarantees.) Thus, at least one server is honest. However, if we use other metadata private messaging systems, as long as there is one honest server, Colo will still guarantee privacy.

B.1.1 Proof for the three phases

We clarify that since Colo reveals aggregated results directly to the analyst, Colo itself is vulnerable to attacks where the adversary may get information by simply inspecting the final result. That is, even if all devices and servers are honest, the final result itself may reveal information. Though we consider this out of the scope of Colo, Colo can be extended to add differential privacy (DP) [53, 42, 54], by asking the servers to add DP noise before revealing the final result.

In light of revealing the final result, we will show the following in this section: i) Colo will only reveal the final aggregated result to the analyst and any other information will not be leaked; ii) in the presence of a malicious adversary, an honest device's input can change the final result by at most the same bound as the case where all parties are honest. These two properties enable Colo to provide provable privacy guarantees.

In order to show the above, we define a concept of *sensitivity*. Recall that in local aggregation phase (Figure 4.4), two neighboring devices $self$ and $neighbor$ will evaluate a function F. We define the sensitivity of F to be the maximum change in output of F if $self$ or $neighbor$'s input changes.

Assumption B.1.4 *Say the query requires the two neighboring devices to evaluate F together and the output range of F is $[0, B]$. We assume there is a local aggregation protocol that can enable two neighboring devices to evaluate a function F' that has the same sensitivity as F, without leaking any information about honest devices.*

For now we assume such a local aggregation protocol exists and later we will prove our local aggregation satisfies this requirement. We now need to prove that if the local aggregation evaluates a (different) function F', which has the same sensitivity as the function F given by the query, the global aggregation phase won't leak information more than the final result.

Claim B.1.1 *If Assumption B.1.4 is true, the global aggregation will not leak information more than the final result and each honest device's input can cause a change to the final result by at most $B \cdot DegreeBound$.*

Proof:

Recall that in the global aggregation phase, the local results are secret-shared with the servers (§4.3.4). Based on the Assumption B.1.3, we know there is at least one honest server and thus at least one share remains unknown to the adversary. So the local results will not be known to the adversary. The only thing the adversary can learn about is the final result (global aggregated result).

Next, we will show that one honest device's input can change the global result by at most $B \cdot DegreeBound$

For any pair of neighboring devices $(self, neighbor)$, by Assumption B.1.4, they will get $F'(self, neighbor)$ as the local result, and by changing $self$ or $neighbor$'s inputs, the local result $F'(self, neighbor)$ will change by at most B, which is the sensitivity of F.

In the global aggregation, again since at least one server is honest, it will include its share of each (honest) device's local result exactly once. This ensures that $F'(self, neighbor)$ will be included at most once in the global aggregation.

Since an honest device has at most $DegreeBound$ of neighbors and for each of them the honest device can change the global result by at most B, each honest device's input can change the final global result by at most $B \cdot DegreeBound$.

■

The above claim shows that in Colo only the final aggregate result will be leaked and each honest device's input can change the final result by some bound, which is the same as the case when all devices are honest-but-curious.

B.1.2 Local aggregation

In this section, we will prove our local aggregation protocol meets Assumption B.1.4, by first showing no information will be leaked about honest devices, and then discussing the sensitivity.

Claim B.1.2 *The local aggregation protocol in Figure 4.4 will not leak any information about honest devices' data.*

Proof: First, the topology data is protected by Assumption B.1.2, since no one can observe the communication between two devices.

Second, we consider a malicious $self$ device that wants to compromise an honest $neighbor$ device's data privacy. The malicious $self$ device receives commitments and ZKP (line (4) in Figure 4.4). The underlying cryptographic primitives ensure that nothing will be leaked (commitments and ZKP don't reveal their inputs). Later, the malicious $self$ device participates in the oblivious transfers (line (5) in Figure 4.4). The maliciously secure OT ensures that the $self$ device can only learn at most one entry of T and this entry is further protected by a random mask only known to the $neighbor$ device. So no information is leaked to a malicious $self$ device.

Third, we consider a malicious $neighbor$ device. The $self$ device will only run OT with the $neighbor$ device and the maliciously secure OT (that protects against a malicious sender, that is, $neighbor$) ensures that this choice number won't be known to the $neighbor$ device.

■

The above shows that no information about honest devices will be leaked during our local aggregation protocol.

Next we need to show that the output of the local aggregation is bounded by B.

Claim B.1.3 *The sensitivity of the local aggregation protocol will be bounded by B.*

Proof: Note that we do not need to consider attacks that impact correctness but not privacy. For example, the $neighbor$ device adds 10^5 to $-r$ locally or simply sets $-r$ to a random number. These attacks will not increase the sensitivity of the local computation since

it doesn't depend on honest devices' input. So we will show that $T[i]$, namely shares $-r$ held by the $neighbor$ device and $T[i] + r$ held by the $self$ device, will be in the range of $[0, B]$.

If the $neighbor$ device is honest, it wants to ensure that the output is bounded by B, because otherwise, its privacy will be impacted. The $neighbor$ device can easily make sure the output is bounded by B because it constructs the array T and the malicious $self$ device cannot interfere with this process.

If a $self$ device is honest, it also wants to ensure the bound, since a malicious $neighbor$ device may provide a malicious T with some entries to be much greater than B, e.g., 10^6. This is achieved with the help of commitments and ZKP. The $self$ device can verify that the entry it receives via OT is the one committed in the commitment, which is ensured by the collision-resistantance property of the commitment scheme. Further, the ZKP ensures all the committed values equal the same mask $-r$ with some numbers in $[0, B]$ added to it.

■

B.2 Mycelium cost calculations

Mycelium is a system designed for federated graph analytics in a strong threat model and is focused on local aggregation. For global aggregation, Mycelium uses Orchard [28]. Both systems have only partial implementations available. We didn't benchmark Mycelium for two reasons. First, only partial implementation is available, and implementing the full protocol is hard with existing tools (as we will explain below). Second, we use Mycelium as a baseline for the purpose of situating Colo's costs and showing that federated graph analytics in a strong threat model is a hard problem; for this purpose, the lower-bound estimates would be enough.

Device-side CPU. For a Mycelium device, one dominant cost is proving multiplications of 10 BGV ciphertexts. Mycelium's design is to follow re-linearization of ciphertexts, but with some differences. Mycelium assumes a degree bound of 10, and each device multiplies 11 ciphertexts (10 neighbors and itself) together to get a 12-poly ciphertext, without running any re-linearization locally. Later, after the global aggregation of all 12-poly ciphertexts (across devices), the aggregator runs re-linearization 10 times, to get back a 2-poly ciphertext.

Mycelium's public code of this ZKP circuit is not complete due to several reasons: i) their circuit is *point-wise* polynomial multiplication but not polynomial multiplications. That is, their output of 10 ciphertext multiplications is still a 2-poly ciphertext, which doesn't match their description; ii) their implementation doesn't consider the difference between the ZKP field (they use ZoKrates [165]) and the BGV field. That is, they assume the BGV field to be the same as the ZKP field. More precisely, they use a 550-bit field for BGV, but ZoKrates only supports curves BN128, BLS12-381, BLS12-377, and BW6-76, and none of them have a 550-bit field. The largest ZKP field, BW6-761, is 377-bit.

We are not able to implement a circuit following their design of delaying re-linearization due to the large number of inputs and outputs to the circuit. We used ark_groth16 [90] to prove polynomial multiplications but quickly went beyond what ark_groth16 can support for the first two or three multiplications since each polynomial has 32,768 dimensions. Also, the

ZKP field size for BW6-761 is a 377-bit number, and proving mod 550-bit operations on a 377-bit field was not feasible.

Given these issues in Mycelium's implementation, we take numbers from a recent paper [227] (Table 2) that performs and proves BGV multiplications. Specifically, their circuit proves one multiplication between two ciphertexts and maintains the number of polynomials in the ciphertext. Their FHE scheme assumes a dimension of 8192 and a field size of 137-bit, which is far smaller than Mycelium's dimension of 32768 and a field size of 550-bit. So we take the 633 s proving time from [227] as a lower-bound estimate for CPU cost to prove one ciphertext multiplication in Mycelium.

Server-side CPU. For the server-side CPU, we use their released scripts that estimate server-side costs. However, we only include the server-side CPU cost for local aggregation since their global aggregation system Orchard also has only a partial implementation available. Thus, again server CPU is a lower-bound estimate.

Network overhead. Finally for the network cost, for both the devices and the server, we take both the local aggregation and global aggregation into account. Mycelium (and Orchard) only have device-server traffic, that is, they route all messages between devices through the servers, and thus the server-side cost is simply a sum of all the devices' network cost. For the local aggregation, we use their released code to estimate this cost, for both the devices and the server. Specifically, their released script estimates the server's average outbound network per device. For example, when the degree bound is 10 (default setting for Mycelium), the output from the script matches the 350 MiB sent traffic reported in their paper. This actually means 700 MiB of network transfer per device since in Mycelium the server plays as a mailbox and is only responsible for forwarding messages between devices. So on average, the server sends 350 MiB to one device, which also means on average the server receives 350 MiB from one device, which translates to about 700 MiB per device. For global aggregation, we use the 19 ciphertexts cost reported in Orchard (1 to upload and 18 to verify the server's aggregation). As specified in Mycelium, each ciphertext is a 52-poly ciphertext for a degree bound of 50. That is, each ciphertext is about 111.8 MiB.

An example. Here, we will give an example to show how we estimate the costs when there are 1 million devices and the degree bound is 50. For device-side CPU, as we said, proving one multiplication requires 633 s CPU time and thus for 50 neighbors (with itself), there are 50 multiplications, leading to 8.79 hour CPU time. For device-side network, we use their released script to estimate the network sent for local aggregation, which is 1.805 GiB, thus 3.61 GiB for sent and received. The global aggregation is 19 ciphertexts transferred, or 2.12 GiB. So in total, the device-side network cost is 5.73 GiB. For the server-side CPU, we run their script to estimate the server-side CPU for the default degree bound 10 setting, that is, 1305 hour. Finally, for the server-side network, since in Mycelium, there is only device-server network (we didn't include other devices like committee devices), we just need to multiply the device-side network cost by the total number of devices. This is 5730 TiB.

Appendix C

Virtual pooling supplementary

C.1 Detailed Cox model computation

Assume there are K health systems and health system i has data (X_i, T_i, E_i). We assume that there are d death events, and we denote their timestamps are $T_1^D < T_2^D < ... < T_d^D$. We use the following notation:

- $X_{i[l]}$ to represent the individual at health system i and index l.
- $D_{i(j)}$ to represent indices of individuals who experience death at T_j^D.
- $R_{i(j)}$ to represent indices of individuals who are at risk at timestamp T_j^D at site i.
- D_i to represent indices of individuals who experience death at any timestamp.

The detailed protocol description can be found in the Figure C.1.

Cox model phase of virtual pooling (without TIE optimization)

1. The coordinator $\mathcal{A}$ initial $\beta = \vec{0}$ and repeat Newton-Raphson steps (2 - 8) till convergence.

One Newton-Raphson step

2. Each health system i computes, for each timestamp j, $tie\_\phi_{i(j)} = \sum_{l \in D_{i(j)}} \exp(\beta^T X_{i[l]})$ and $risk\_\phi_{i(j)} = \sum_{l \in R_{i(j)}} \exp(\beta^T X_{i[l]})$, and share them with the coordinator.
3. The coordinator combines to get $tie\_\phi_{(j)} = \sum_i tie\_\phi_{i(j)}$ and $risk\_\phi_{(j)} = \sum_i risk\_\phi_{i(j)}$ for every timestamp j, and make them public to health systems.
4. Each health system i computes local share of gradient $g_i = \sum_{l \in D_i} X_{i[j]} - \sum_j \frac{\sum_{l \in R_{i(j)}} X_{i[l]} \exp(\beta^T X_{i[l]})}{risk\_\phi_{(j)}}$ and local share of first term of Hessian $ha_i = -\sum_j \frac{\sum_{l \in R_{i(j)}} X_{i[l]} X_{i[l]}^T \exp(\beta^T X_{i[l]})}{risk\_\phi_{(j)}}$.
5. Each health system i computes its local share of $A_i = \frac{\sum_{l \in R_{i(j)}} X_{i[l]} \exp(\beta^T X_{i[l]})}{risk\_\phi_{(j)}}$.
6. Each health system generates shamir secret sharing of $[A_i] \leftarrow Shamir(A_i)$ and send shares $[A_i]$ to other health systems via the coordinator (using symmetric encryption).
7. After receiving shares from all other health systems, health systems locally compute $[A] = \sum_i [A_i]$ and compute $[AA^T]$ by shamir secret sharing.
8. Each health system shares g_i, ha_i and its local share of $[AA^T]$ with the coordinator.
9. The coordinator combines shares of $[AA^T]$ to reconstruct AA^T and compute $g = \sum g_i$, $H = \sum ha_i + AA^T$. The coordinator update $\beta = \beta - H^{-1}g$

P value

8. After the Cox model converges, we estimate $var(\beta)$ to be diagonal of H^{-1} and we apply Wald test to compute $z = \frac{\beta - \vec{0}}{\sqrt{var(\beta)}}$, and compute P value based on Z value.

Figure C.1: Virtual pooling's Cox model phase.

Bibliography

[1] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, *Federated learning for mobile keyboard prediction*, *arXiv preprint arXiv:1811.03604* (2018).

[2] K. M. Andersen, B. A. Bates, E. S. Rashidi, A. L. Olex, R. B. Mannon, R. C. Patel, J. Singh, J. Sun, P. G. Auwaerter, D. K. Ng, *et. al.*, *Long-term use of immunosuppressive medicines and in-hospital COVID-19 outcomes: a retrospective cohort study using data from the National COVID Cohort Collaborative*, *The Lancet Rheumatology* (2022).

[3] "Anthem is warning consumers about its huge data breach." https://www.latimes.com/business/la-fi-mh-anthem-is-warning-consumers-20150306-column.html, 2015.

[4] "Premera Blue Cross Says Data Breach Exposed Medical Data." https://www.nytimes.com/2015/03/18/business/premera-blue-cross-says-data-breach-exposed-medical-data.html, 2015.

[5] I. Vojinovic, "Data breach statistics that will make you think twice before filling out an online form." https://dataprot.net/statistics/data-breach-statistics/.

[6] "Recent Cyber Attacks & Data Breaches In 2023." https://purplesec.us/security-insights/data-breaches/.

[7] D. Ramage, "Federated analytics: Collaborative data science without data collection." `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`, 2020.

[8] L. Zhu, Z. Liu, and S. Han, *Deep leakage from gradients*, in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[9] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, *Exploiting unintended feature leakage in collaborative learning*, in *IEEE Symposium on Security and Privacy (S&P)*, pp. 691–706, 2019.

[10] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, *Membership inference attacks against machine learning models*, in *IEEE Symposium on Security and Privacy (S&P)*, pp. 3–18, 2017.

[11] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, *Local privacy and statistical minimax rates*, in *Symposium on Foundations of Computer Science (FOCS)*, 2013.

[12] Ú. Erlingsson, V. Pihur, and A. Korolova, *RAPPOR: Randomized aggregatable privacy-preserving ordinal response*, in *ACM Conference on Computer and Communications Security (CCS)*, pp. 1054–1067, 2014.

[13] L. Sun, J. Qian, and X. Chen, *LDP-FL: Practical private aggregation in federated learning with local differential privacy*, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1571–1578, 2021.

[14] L. He, S. P. Karimireddy, and M. Jaggi, *Secure byzantine-robust machine learning*, *arXiv preprint arXiv:2006.04747* (2020).

[15] M. A. Pathak, S. Rane, and B. Raj, *Multiparty differential privacy via aggregation of locally trained classifiers*, in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1876–1884, 2010.

[16] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, *LDP-Fed: Federated learning with local differential privacy*, in *Proceedings of the ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 61–66, 2020.

[17] D. Li, W. Lu, D. Shu, S. Toh, and R. Wang, *Distributed cox proportional hazards regression using summary-level information*, *Biostatistics* (2023).

[18] C. Luo, R. Duan, A. C. Naj, H. R. Kranzler, J. Bian, and Y. Chen, *ODACH: a one-shot distributed algorithm for cox model with heterogeneous multi-center data*, *Scientific reports* (2022).

[19] N. Hynes, R. Cheng, and D. Song, *Efficient deep learning on multi-source private data*, *arXiv preprint arXiv:1807.06689* (2018).

[20] S. Fei, Z. Yan, W. Ding, and H. Xie, *Security vulnerabilities of SGX and countermeasures: A survey*, *ACM Computing Surveys* (2021).

[21] G. Beniamini, "Trust issues: Exploiting trustzone TEEs.." `https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html`. Accessed: 2022-01-30.

[22] M. Li, Y. Zhang, Z. Lin, and Y. Solihin, *Exploiting unprotected I/O operations in AMD's secure encrypted virtualization*, in *USENIX Security Symposium*, 2019.

[23] Y. Aono, T. Hayashi, L. Wang, S. Moriai, *et. al.*, *Privacy-preserving deep learning via additively homomorphic encryption*, *IEEE Transactions on Information Forensics and Security* (2017) 1333–1345.

[24] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, *ELSA: Secure aggregation for federated learning with malicious actors*, *Cryptology ePrint Archive* (2022).

[25] Y. Dong, X. Chen, L. Shen, and D. Wang, *EaSTFLy: Efficient and secure ternary federated learning*, *Computers & Security* (2020) 101824.

[26] A. Fu, X. Zhang, N. Xiong, Y. Gao, H. Wang, and J. Zhang, *VFL: a verifiable federated learning with privacy-preserving for big data in industrial IoT*, *IEEE Transactions on Industrial Informatics* (2020).

[27] M. Jiang, T. Jung, R. Karl, and T. Zhao, *Federated dynamic GNN with secure aggregation*, *arXiv preprint arXiv:2009.07351* (2020).

[28] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce, *Orchard: Differentially private analytics at scale*, in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 1065–1081, 2020.

[29] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen, *Honeycrisp: Large-scale differentially private aggregation without a trusted core*, in *ACM Symposium on Operating Systems Principles (SOSP)*, pp. 196–210, 2019.

[30] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen, *Mycelium: Large-scale distributed graph queries with differential privacy*, in *ACM Symposium on Operating Systems Principles (SOSP)*, 2021.

[31] F. Hartmann, S. Suh, A. Komarzewski, T. D. Smith, and I. Segall, *Federated learning for ranking browser history suggestions*, *arXiv preprint arXiv:1911.11807* (2019).

[32] P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu, *Practical and private (deep) learning without sampling or shuffling*, in *International Conference on Machine Learning*, pp. 5213–5225, 2021.

[33] B. McMahan and A. Thakurta, "Federated learning with formal differential privacy guarantees." `https://ai.googleblog.com/2022/02/federated-learning-with-formal.html`. Accessed: 2022-12-12.

[34] Y. J. Park, Y. J. Choe, O. Park, S. Y. Park, Y.-M. Kim, J. Kim, S. Kweon, Y. Woo, J. Gwack, S. S. Kim, *et. al.*, *Contact tracing during coronavirus disease outbreak, South Korea, 2020*, *Emerging infectious diseases* (2020).

[35] R. Laxminarayan, B. Wahl, S. R. Dudala, K. Gopal, C. Mohan B, S. Neelima, K. Jawahar Reddy, J. Radhakrishnan, and J. A. Lewnard, *Epidemiology and transmission dynamics of COVID-19 in two Indian states*, *Science* (2020).

[36] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, *Learning differentially private recurrent language models*, in *International Conference on Learning Representations*, 2018.

[37] K. Liu and T. Gupta, *Federated learning with differential privacy and an untrusted aggregator*, in *Proceedings of the 10th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, 2024.

[38] K. Liu, R. Wadaskar, and T. Gupta, *Towards an efficient system for differentially-private, cross-device federated learning*, in *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, pp. 7–9, 2021.

[39] K. Liu and T. Gupta, *Federated learning with differential privacy and an untrusted aggregator*, *arXiv preprint arXiv:2312.10789* (2023).

[40] K. Liu and T. Gupta, *Making privacy-preserving federated graph analytics practical (for certain queries)*, in *Proceedings of the 29th ACM Symposium on Access Control Models and Technologies*, 2024.

[41] I. Ahmad, K. Liu, V. Rudrapatna, and T. Gupta, *Virtual Pooling: A method for privacy-preserving statistical analysis and machine learning over distributed networks of healthcare data*, in *Poster in UC Health Conference*, 2024.

[42] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating noise to sensitivity in private data analysis*, in *Theory of Cryptography Conference (TCC)*, pp. 265–284, 2006.

[43] A. Shamir, *How to share a secret*, *Communications of the ACM* **22** (1979), no. 11 612–613.

[44] T. Chou and C. Orlandi, *The simplest protocol for oblivious transfer*, in *International Conference on Cryptology and Information Security in Latin America*, 2015.

[45] J. Fan and F. Vercauteren, *Somewhat practical fully homomorphic encryption.*, *IACR Cryptol. ePrint Arch.* (2012) 144.

[46] Z. Brakerski, *Fully homomorphic encryption without modulus switching from classical GapSVP*, in *Advances in Cryptology—CRYPTO*, pp. 868–886, 2012.

[47] J. Groth, *On the size of pairing-based non-interactive arguments*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 305–326, 2016.

[48] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, in *Artificial intelligence and statistics*, pp. 1273–1282, 2017.

[49] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, *Oort: Efficient federated learning via guided participant selection*, in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 19–35, 2021.

[50] F. Lai, Y. Dai, S. S. Singapuram, J. Liu, X. Zhu, H. V. Madhyastha, and M. Chowdhury, *FedScale: Benchmarking model and system performance of federated learning at scale*, in *International Conference on Machine Learning (ICML)*, 2022.

[51] C. He, S. Li, J. So, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, *Fedml: A research library and benchmark for federated machine learning*, *Advances in Neural Information Processing Systems, Best Paper Award at Federate Learning Workshop* (2020).

[52] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, *HybridAlpha: An efficient approach for privacy-preserving federated learning*, in *ACM Workshop on Artificial Intelligence and Security*, pp. 13–23, 2019.

[53] C. Dwork, *A firm foundation for private data analysis*, *Communications of the ACM* (2011) 86–95.

[54] C. Dwork, A. Roth, *et. al.*, *The algorithmic foundations of differential privacy.*, *Foundations and Trends in Theoretical Computer Science* (2014) 9(3–4):211–407.

[55] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy*, in *ACM Conference on Computer and Communications Security (CCS)*, pp. 308–318, 2016.

[56] S. J. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečnỳ, S. Kumar, and H. B. McMahan, *Adaptive federated optimization*, in *International Conference on Learning Representations*, 2020.

[57] A. Grafberger, M. Chadha, A. Jindal, J. Gu, and M. Gerndt, *FedLess: Secure and scalable federated learning using serverless computing*, in *IEEE International Conference on Big Data (Big Data)*, pp. 164–173, 2021.

[58] pytorch, "Pytorch." https://github.com/pytorch/pytorch.

[59] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, *EMNIST: Extending MNIST to handwritten letters*, in *International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926, 2017.

[60] S. Tan, B. Knott, Y. Tian, and D. J. Wu, *Cryptgpu: Fast privacy-preserving machine learning on the gpu*, in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1021–1038, IEEE, 2021.

[61] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, *Crypten: Secure multi-party computation meets machine learning*, in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[62] M. Naseri, J. Hayes, and E. D. Cristofaro, *Local and central differential privacy for robustness and privacy in federated learning*, *Proceedings of the Network and Distributed System Security Symposium* (2022).

[63] A. C. Yao, *Protocols for secure computations*, in *Annual Symposium on Foundations of Computer Science (SFCS)*, pp. 160–164, 1982.

[64] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, in *ACM Symposium on Theory of Computing (STOC)*, p. 218–229, 1987.

[65] KU Leuven COSIC, "SCALE-MAMBA." https://github.com/KULeuven-COSIC/SCALE-MAMBA.

[66] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, *Multiparty computation from somewhat homomorphic encryption*, in *Advances in Cryptology—CRYPTO*, pp. 643–662, 2012.

[67] S. D. Gordon, D. Starin, and A. Yerukhimovich, *The more the merrier: reducing the cost of large scale mpc*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2021.

[68] A. Ben-Efraim, K. Cong, E. Omri, E. Orsini, N. P. Smart, and E. Soria-Vazquez, *Large scale, actively secure computation from lpn and free-xor garbled circuits*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2021.

[69] R. L. Rivest, L. Adleman, M. L. Dertouzos, *et. al.*, *On data banks and privacy homomorphisms*, *Foundations of secure computation* (1978).

[70] S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, p. 203–225. Association for Computing Machinery, 2019.

[71] A. Roth and T. Roughgarden, *Interactive privacy via the median mechanism*, in *ACM Symposium on Theory of Computing (STOC)*, 2010.

[72] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan, *On the complexity of differentially private data release: efficient algorithms and hardness results*, in *ACM Symposium on Theory of Computing (STOC)*, 2009.

[73] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, *Algorand: Scaling byzantine agreements for cryptocurrencies*, in *ACM Symposium on Operating Systems Principles (SOSP)*, p. 51–68, 2017.

[74] V. Gupta and K. Gopinath, *An extended verifiable secret redistribution protocol for archival systems*, in *International Conference on Availability, Reliability and Security (ARES)*, pp. 8–pp, 2006.

[75] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, *A hybrid approach to privacy-preserving federated learning*, in *ACM Workshop on Artificial Intelligence and Security*, pp. 1–11, 2019.

[76] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, *Our data, ourselves: Privacy via distributed noise generation*, *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)* (2006) 486.

[77] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, *Journal of the ACM (J. ACM)* **27** (1980), no. 4 701–717.

[78] R. Zippel, *Probabilistic algorithms for sparse polynomials*, in *International Symposium on Symbolic and Algebraic Manipulation*, pp. 216–226, 1979.

[79] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, *Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation*, in *Theory of Cryptography Conference (TCC)*, 2006.

[80] H. Chen, W. Dai, M. Kim, and Y. Song, *Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference*, in *ACM Conference on Computer and Communications Security (CCS)*, pp. 395–412, 2019.

[81] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, *Multiparty computation with low communication, computation and interaction via threshold FHE*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 483–501, 2012.

[82] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov, *Opacus: User-friendly differential privacy library in PyTorch*, *arXiv preprint arXiv:2109.12298* (2021).

[83] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, *et. al.*, *Learning algorithms for classification: A comparison on handwritten digit recognition*, *Neural Networks: the statistical mechanics perspective* (1995).

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, *Advances in neural information processing systems* **25** (2012).

[85] A. Krizhevsky, G. Hinton, *et. al.*, *Learning multiple layers of features from tiny images*, .

[86] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[87] P. Warden, *Speech commands: A dataset for limited-vocabulary speech recognition*, *arXiv preprint arXiv:1804.03209* (2018).

[88] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, *arXiv preprint arXiv:1704.04861* (2017).

[89] Microsoft, "Microsoft SEAL (release 3.7)." https://github.com/Microsoft/SEAL.

[90] arkworks, "ark-groth16." https://github.com/arkworks-rs/groth16.

[91] M. Seif, R. Tandon, and M. Li, *Wireless federated learning with local differential privacy*, in *IEEE International Symposium on Information Theory (ISIT)*, pp. 2604–2609, 2020.

[92] A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, and R. Rogers, *Protection against reconstruction and its applications in private federated learning*, *arXiv preprint arXiv:1812.00984* (2018).

[93] M. Hao, H. Li, G. Xu, S. Liu, and H. Yang, *Towards efficient and privacy-preserving federated deep learning*, in *IEEE International Conference on Communications (ICC)*, pp. 1–6, 2019.

[94] L. Sun and L. Lyu, *Federated model distillation with noise-free differential privacy*, *arXiv preprint arXiv:2009.05537* (2020).

[95] T. T. Nguyên, X. Xiao, Y. Yang, S. C. Hui, H. Shin, and J. Shin, *Collecting and analyzing data from smart device users with local differential privacy*, *arXiv preprint arXiv:1606.05053* (2016).

[96] N. Wang, X. Xiao, Y. Yang, J. Zhao, S. C. Hui, H. Shin, J. Shin, and G. Yu, *Collecting and analyzing multidimensional data with local differential privacy*, in *IEEE International Conference on Data Engineering (ICDE)*, pp. 638–649, 2019.

[97] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, *Secure federated submodel learning*, *arXiv preprint arXiv:1911.02254* (2019).

[98] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, *Blockchain and federated learning for privacy-preserved data sharing in industrial IoT*, *IEEE Transactions on Industrial Informatics* (2019) 4177–4186.

[99] X. Chen, J. Ji, C. Luo, W. Liao, and P. Li, *When machine learning meets blockchain: A decentralized, privacy-preserving and secure design*, in *IEEE International Conference on Big Data (Big Data)*, pp. 1178–1187, 2018.

[100] V. Mugunthan, R. Rahman, and L. Kagal, *BlockFLow: An accountable and privacy-preserving solution for federated learning*, *arXiv preprint arXiv:2007.03856* (2020).

[101] C. Chen, J. Zhou, B. Wu, W. Fang, L. Wang, Y. Qi, and X. Zheng, *Practical privacy preserving POI recommendation*, *ACM Transactions on Intelligent Systems and Technology (TIST)* (2020) 1–20.

[102] J. Ding, G. Liang, J. Bi, and M. Pan, *Differentially private and communication efficient collaborative learning*, in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[103] D. Froelicher, P. Egger, J. S. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux, *UnLynx: A decentralized system for privacy-conscious data sharing.*, *Proceedings on Privacy Enhancing Technologies* (2017) 232–250.

[104] D. Zeng, S. Liu, and Z. Xu, *Aggregating gradients in encoded domain for federated learning*, *arXiv preprint arXiv:2205.13216* (2022).

[105] A. G. Sébert, R. Sirdey, O. Stan, and C. Gouy-Pailler, *Protecting data from all parties: Combining FHE and DP in federated learning*, *arXiv preprint arXiv:2205.04330* (2022).

[106] M. Chase, R. Gilad-Bachrach, K. Laine, K. Lauter, and P. Rindal, *Private collaborative neural network learning*, *IACR Cryptol. ePrint Arch.* (2017).

[107] V. Rastogi and S. Nath, *Differentially private aggregation of distributed time-series with transformation and encryption*, in *Proceedings of ACM SIGMOD International Conference on Management of data*, pp. 735–746, 2010.

[108] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near, *Efficient differentially private secure aggregation for federated learning via hardness of learning with errors*, *arXiv preprint arXiv:2112.06872* (2021).

[109] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, S. Kadhe, and H. Ludwig, *DeTrust-FL: Privacy-preserving federated learning in decentralized trust setting*, in *IEEE International Conference on Cloud Computing (CLOUD)*, 2022.

[110] Z. Jiang, W. Wang, and Y. Liu, *Flashe: Additively symmetric homomorphic encryption for cross-silo federated learning*, *arXiv preprint arXiv:2109.00675* (2021).

[111] C. Liu, S. Chakraborty, and D. Verma, *Secure model fusion for distributed learning using partial homomorphic encryption*, in *Policy-Based Autonomic Data Governance*, pp. 154–179. Springer, 2019.

[112] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, *Privacy-preserving federated learning based on multi-key homomorphic encryption*, *arXiv preprint arXiv:2104.06824* (2021).

[113] K. Mandal and G. Gong, *PrivFL: Practical privacy-preserving federated regressions on high-dimensional data over mobile networks*, in *Proceedings of ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp. 57–68, 2019.

[114] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, *BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning*, in *USENIX Annual Technical Conference (USENIX ATC)*, pp. 493–506, 2020.

[115] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, *Poseidon: Privacy-preserving federated neural network learning*, in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2021.

[116] G. Xu, X. Han, S. Xu, T. Zhang, H. Li, X. Huang, and R. H. Deng, *Hercules: Boosting the performance of privacy-preserving federated learning*, *arXiv preprint arXiv:2207.04620* (2022).

[117] C. Beguier, M. Andreux, and E. W. Tramel, *Efficient sparse secure aggregation for federated learning*, *arXiv preprint arXiv:2007.14861* (2020).

[118] Q. Chen, Z. Wang, W. Zhang, and X. Lin, *PPT: A privacy-preserving global model training protocol for federated learning in P2P networks*, *arXiv preprint arXiv:2105.14408* (2021).

[119] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten, *EIFFeL: Ensuring integrity for federated learning*, *arXiv preprint arXiv:2112.12727* (2021).

[120] I. Ergun, H. U. Sami, and B. Guler, *Sparsified secure aggregation for privacy-preserving federated learning*, *arXiv preprint arXiv:2112.12872* (2021).

[121] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame, *et. al.*, *SAFELearn: secure aggregation for private federated learning*, in *IEEE Security and Privacy Workshops (SPW)*, pp. 56–62, 2021.

[122] J. Guo, Z. Liu, K.-Y. Lam, J. Zhao, Y. Chen, and C. Xing, *Secure weighted aggregation for federated learning*, *arXiv preprint arXiv:2010.08730* (2020).

[123] M. Hao, H. Li, G. Xu, H. Chen, and T. Zhang, *Efficient, private and robust federated learning*, in *Annual Computer Security Applications Conference*, pp. 45–60, 2021.

[124] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, *Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning*, in *ICML Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2020.

[125] K. H. Li, P. P. B. de Gusmão, D. J. Beutel, and N. D. Lane, *Secure aggregation for federated learning in flower*, in *Proceedings of ACM International Workshop on Distributed Machine Learning*, pp. 8–14, 2021.

[126] Y. Liu, Z. Ma, X. Liu, S. Ma, S. Nepal, R. H. Deng, and K. Ren, *Boosting privately: Federated extreme gradient boosting for mobile crowdsensing*, in *International Conference on Distributed Computing Systems (ICDCS)*, pp. 1–11, 2020.

[127] J. So, B. Güler, and A. S. Avestimehr, *Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning*, *IEEE Journal on Selected Areas in Information Theory* (2021) 479–489.

[128] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, *VerifyNet: Secure and verifiable federated learning*, *IEEE Transactions on Information Forensics and Security* (2019) 911–926.

[129] S. Zhang, Z. Li, Q. Chen, W. Zheng, J. Leng, and M. Guo, *Dubhe: Towards data unbiasedness with homomorphic encryption in federated learning client selection*, in *International Conference on Parallel Processing*, pp. 1–10, 2021.

[130] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, *PPFL: privacy-preserving federated learning with trusted execution environments*, *arXiv preprint arXiv:2104.14380* (2021).

[131] H. Hashemi, Y. Wang, C. Guo, and M. Annavaram, *Byzantine-robust and privacy-preserving framework for FedML*, *arXiv preprint arXiv:2105.02295* (2021).

[132] D. L. Quoc and C. Fetzer, *SecFL: Confidential federated learning using TEEs*, *arXiv preprint arXiv:2110.00981* (2021).

[133] S. Sav, A. Diaa, A. Pyrgelis, J.-P. Bossuat, and J.-P. Hubaux, *Privacy-preserving federated recurrent neural networks*, *arXiv preprint arXiv:2207.13947* (2022).

[134] H. Corrigan-Gibbs and D. Boneh, *Prio: Private, robust, and scalable computation of aggregate statistics*, in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 259–282, 2017.

[135] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, *Secure single-server aggregation with (poly) logarithmic overhead*, in *ACM Conference on Computer and Communications Security (CCS)*, pp. 1253–1269, 2020.

[136] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, *Practical secure aggregation for privacy-preserving machine learning*, in *ACM Conference on Computer and Communications Security (CCS)*, pp. 1175–1191, 2017.

[137] Z. Liu, S. Chen, J. Ye, J. Fan, H. Li, and X. Li, *DHSA: efficient doubly homomorphic secure aggregation for cross-silo federated learning*, *The Journal of Supercomputing* (2022).

[138] K. Wan, H. Sun, M. Ji, and G. Caire, *Information theoretic secure aggregation with uncoded groupwise keys*, *arXiv preprint arXiv:2204.11364* (2022).

[139] Z. Liu, J. Guo, K.-Y. Lam, and J. Zhao, *Efficient dropout-resilient aggregation for privacy-preserving machine learning*, *IEEE Transactions on Information Forensics and Security* (2022).

[140] I. Damgård and M. Jurik, *A generalisation, a simplification and some applications of Paillier's probabilistic public-key system*, in *Proceedings of International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, pp. 119–136, 2001.

[141] B. Hitaj, G. Ateniese, and F. Perez-Cruz, *Deep models under the GAN: Information leakage from collaborative deep learning*, in *ACM Conference on Computer and Communications Security (CCS)*, p. 603–618, 2017.

[142] "TraceTogether." https://www.tracetogether.gov.sg/.

[143] "CovidSafe." https://covidsafe.cs.washington.edu/.

[144] "SwissCovid." https://www.bag.admin.ch/bag/en/home/krankheiten/ausbrueche-epidemien-pandemien/aktuelle-ausbrueche-epidemien/novel-cov/swisscovid-app-und-contact-tracing.html.

[145] "Centers for Disease Control and Prevention." https://www.cdc.gov/.

[146] D. Günther, M. Holz, B. Judkewitz, H. Möllering, B. Pinkas, T. Schneider, and A. Suresh, *Privacy-preserving epidemiological modeling on mobile graphs*, *arXiv preprint arXiv:2206.00539* (2022).

[147] D. Günther, M. Holz, B. Judkewitz, H. Möllering, B. Pinkas, T. Schneider, and A. Suresh, *Poster: Privacy-preserving epidemiological modeling on mobile graphs*, in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[148] D. Lazar, Y. Gilad, and N. Zeldovich, *Karaoke: Distributed private messaging immune to passive traffic analysis*, in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 711–725, 2018.

[149] D. Lazar, Y. Gilad, and N. Zeldovich, *Yodel: Strong metadata security for voice calls*, in *ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[150] A. C. Yao, *How to generate and exchange secrets*, in *Annual Symposium on Foundations of Computer Science (SFCS)*, 1986.

[151] M. O. Rabin, *How to exchange secrets with oblivious transfer*, *Cryptology ePrint Archive* (2005).

[152] G. Brassard, D. Chaum, and C. Crépeau, *Minimum disclosure proofs of knowledge*, *Journal of computer and system sciences* (1988).

[153] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, *Poseidon: A new hash function for Zero-Knowledge proof systems*, in *USENIX Security Symposium*, 2021.

[154] J. Groth, *On the size of pairing-based non-interactive arguments*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 305–326, 2016.

[155] B. Nikolay, H. Salje, M. J. Hossain, A. D. Khan, H. M. Sazzad, M. Rahman, P. Daszak, U. Ströher, J. R. Pulliam, A. M. Kilpatrick, *et. al.*, *Transmission of Nipah virus—14 years of investigations in Bangladesh*, *New England Journal of Medicine* (2019).

[156] J. Mossong, N. Hens, M. Jit, P. Beutels, K. Auranen, R. Mikolajczyk, M. Massari, S. Salmaso, G. S. Tomba, J. Wallinga, *et. al.*, *Social contacts and mixing patterns relevant to the spread of infectious diseases*, *PLoS medicine* (2008).

[157] Q.-L. Jing, M.-J. Liu, Z.-B. Zhang, L.-Q. Fang, J. Yuan, A.-R. Zhang, N. E. Dean, L. Luo, M.-M. Ma, I. Longini, *et. al.*, *Household secondary attack rate of COVID-19 and associated determinants in Guangzhou, China: A retrospective cohort study*, *The Lancet Infectious Diseases* (2020).

[158] D. F. Gudbjartsson, A. Helgason, H. Jonsson, O. T. Magnusson, P. Melsted, G. L. Norddahl, J. Saemundsdottir, A. Sigurdsson, P. Sulem, A. B. Agustsdottir, *et. al.*, *Spread of SARS-CoV-2 in the Icelandic population*, *New England Journal of Medicine* (2020).

[159] L. Danon, J. M. Read, T. A. House, M. C. Vernon, and M. J. Keeling, *Social encounter networks: Characterizing Great Britain*, *Proceedings of the Royal Society B: Biological Sciences* (2013).

[160] Q. Bi, Y. Wu, S. Mei, C. Ye, X. Zou, Z. Zhang, X. Liu, L. Wei, S. A. Truelove, T. Zhang, *et. al.*, *Epidemiology and transmission of COVID-19 in 391 cases and 1286 of their close contacts in Shenzhen, China: A retrospective cohort study*, *The Lancet infectious diseases* (2020).

[161] D. C. Adam, P. Wu, J. Y. Wong, E. H. Lau, T. K. Tsang, S. Cauchemez, G. M. Leung, and B. J. Cowling, *Clustering and superspreading potential of SARS-CoV-2 infections in Hong Kong*, *Nature Medicine* (2020).

[162] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, *GraphX: Graph processing in a distributed dataflow framework*, in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[163] G. R. Blakley, *Safeguarding cryptographic keys*, in *Workshop on Managing Requirements Knowledge (MARK)*, 1979.

[164] M. Yagisawa, *Fully homomorphic encryption without bootstrapping*, *Cryptology ePrint Archive* (2015).

[165] "ZoKrates." https://zokrates.github.io/.

[166] J. Gehrke, E. Lui, and R. Pass, *Towards privacy for social networks: A zero-knowledge based definition of privacy*, in *Theory of Cryptography Conference (TCC)*, 2011.

[167] Y. Li, M. Purcell, T. Rakotoarivelo, D. Smith, T. Ranbaduge, and K. S. Ng, *Private graph data release: A survey*, *ACM Computing Surveys* (2023).

[168] W.-Y. Day, N. Li, and M. Lyu, *Publishing graph degree distribution with node differential privacy*, in *ACM SIGMOD*, 2016.

[169] X. Ding, X. Zhang, Z. Bao, and H. Jin, *Privacy-preserving triangle counting in large graphs*, in *The Conference on Information and Knowledge Management (CIKM)*, 2018.

[170] Y. Lindell and B. Pinkas, *An efficient protocol for secure two-party computation in the presence of malicious adversaries*, in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2007.

[171] P. Mohassel and B. Riva, *Garbled circuits checking garbled circuits: More efficient and secure two-party computation*, in *Advances in Cryptology—CRYPTO*, 2013.

[172] Y. Lindell and B. Pinkas, *Secure two-party computation via cut-and-choose oblivious transfer*, *Journal of Cryptology* (2012).

[173] M. Orrù, E. Orsini, and P. Scholl, "Actively secure 1-out-of-N OT extension with application to private set intersection." Cryptology ePrint Archive, Paper 2016/933, 2016.

[174] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, *Bulletproofs: Short proofs for confidential transactions and more*, in *IEEE Symposium on Security and Privacy (S&P)*, pp. 315–334, 2018.

[175] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, *Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger*, *IEEE Access* **10** (2022) 42067–42082.

[176] D. Mansy and P. Rindal, *Endemic oblivious transfer*, in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 309–326, 2019.

[177] L. Roy, "SoftSpokenOT: Communication-computation tradeoffs in OT extension." Cryptology ePrint Archive, Paper 2022/192, 2022.

[178] A. C. Yao, *Theory and application of trapdoor functions*, in *Annual Symposium on Foundations of Computer Science (SFCS)*, 1982.

[179] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo random bits*, in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pp. 227–240, 2019.

[180] D. J. Bernstein, *ChaCha, a variant of Salsa20*, in *Workshop record of SASC*, 2008.

[181] D. L. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*, *Communications of the ACM* **24** (1981), no. 2 84–90.

[182] D. Charousset, R. Hiesgen, and T. C. Schmidt, *Revisiting Actor Programming in C++*, *Computer Languages, Systems & Structures* **45** (April, 2016) 105–131.

[183] "Neptune." https://github.com/lurk-lab/neptune.

[184] L. R. Peter Rindal, "libOTe: an efficient, portable, and easy to use Oblivious Transfer Library." https://github.com/osu-crypto/libOTe.

[185] "libsodium." https://github.com/jedisct1/libsodium.

[186] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June, 2014.

[187] K. Newatia, "Mycelium." https://github.com/karannewatia/Mycelium.

[188] AWS, "Compute Savings Plans - Amazon Web Services." https://aws.amazon.com/savingsplans/compute-pricing/.

[189] S. Bowe, A. Gabizon, and I. Miers, *Scalable multi-party computation for zk-SNARK parameters in the random beacon model*, *Cryptology ePrint Archive* (2017).

[190] Azure, "Pricing - Bandwidth | Microsoft Azure." https://azure.microsoft.com/en-us/pricing/details/bandwidth/.

[191] AWS, "Amazon EC2 On-Demand Pricing." https://aws.amazon.com/ec2/pricing/on-demand/.

[192] X. Li, F. Li, and M. Gao, *Flare: A fast, secure, and memory-efficient distributed analytics framework*, *International Conference on Very Large Data Bases (VLDB)* (2023).

[193] M. Du, S. Wu, Q. Wang, D. Chen, P. Jiang, and A. Mohaisen, *GraphShield: Dynamic large graphs for secure queries with forward privacy*, *IEEE Transactions on Knowledge and Data Engineering* **34** (2022), no. 7 3295–3308.

[194] P. Xie and E. Xing, *CryptGraph: Privacy preserving graph analytics on encrypted graph*, *arXiv preprint arXiv:1409.5021* (2014).

[195] S. Wang, Y. Zheng, X. Jia, and X. Yi, *Pegraph: A system for privacy-preserving and efficient search over encrypted social graphs*, *IEEE Transactions on Information Forensics and Security* (2022).

[196] S. Lai, X. Yuan, S.-F. Sun, J. K. Liu, Y. Liu, and D. Liu, *Graphse^2: An encrypted graph database for privacy-preserving social search*, in *ACM ASIA Conference on Computer and Communications Security (CCS)*, 2019.

[197] C. Liu, L. Zhu, X. He, and J. Chen, *Enabling privacy-preserving shortest distance queries on encrypted graph data*, *IEEE Transactions on Dependable and Secure Computing* (2018).

[198] Y. Luo, D. Wang, S. Fu, M. Xu, Y. Chen, and K. Huang, *Approximate shortest distance queries with advanced graph analytics over large-scale encrypted graphs*, in *International Conference on Mobility, Sensing and Networking (MSN)*, 2022.

[199] S. Mazloom and S. D. Gordon, *Secure computation with differentially private access patterns*, in *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[200] S. Mazloom, P. H. Le, S. Ranellucci, and S. D. Gordon, *Secure parallel computation on national scale volumes of data*, in *USENIX Security Symposium*, 2020.

[201] S. Wang, Y. Zheng, X. Jia, and X. Yi, *Privacy-preserving analytics on decentralized social graphs: The case of eigendecomposition*, *IEEE Transactions on Knowledge and Data Engineering* (2022).

[202] S. Sharma, J. Powers, and K. Chen, *PrivateGraph: Privacy-preserving spectral analysis of encrypted graphs in the cloud*, *IEEE Transactions on Knowledge and Data Engineering* (2018).

[203] S. Wang, Y. Zheng, X. Jia, H. Huang, and C. Wang, *OblivGM: Oblivious attributed subgraph matching as a cloud service*, *IEEE Transactions on Information Forensics and Security* (2022).

[204] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, *GraphSC: Parallel secure computation made easy*, in *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[205] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, *Secure graph analysis at scale*, in *ACM Conference on Computer and Communications Security (CCS)*, 2021.

[206] S. Wang, Y. Zheng, X. Jia, Q. Wang, and C. Wang, *Mago: Maliciously secure subgraph counting on decentralized social graphs*, *IEEE Transactions on Information Forensics and Security* (2023).

[207] F. D. McSherry, *Privacy integrated queries: An extensible platform for privacy-preserving data analysis*, in *ACM SIGMOD*, 2009.

[208] A. Narayan and A. Haeberlen, *DJoin: Differentially private join queries over distributed databases*, in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[209] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers, *Shrinkwrap: efficient SQL query processing in differentially private data federations*, *International Conference on Very Large Data Bases (VLDB)* (2018).

[210] F. Han, L. Zhang, H. Feng, W. Liu, and X. Li, *Scape: Scalable collaborative analytics system on private database with malicious security*, in *IEEE International Conference on Data Engineering (ICDE)*, 2022.

[211] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers, *SMCQL: Secure querying for federated databases*, *International Conference on Very Large Data Bases (VLDB)* (2017).

[212] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein, *Senate: A maliciously-secure MPC platform for collaborative analytics*, in *USENIX Security Symposium*, 2021.

[213] A. Papadimitriou, A. Narayan, and A. Haeberlen, *Dstress: Efficient differentially private computations on distributed data*, in *ACM European Conference on Computer Systems (EuroSys)*, 2017.

[214] D. R. Cox, *Regression models and life-tables*, *Journal of the Royal Statistical Society: Series B (Methodological)* (1972).

[215] D. E. Ho, K. Imai, G. King, and E. A. Stuart, *MatchIt: Nonparametric preprocessing for parametric causal inference*, *Journal of Statistical Software* **42** (2011), no. 8 1–28.

[216] T. M. Therneau, *A Package for Survival Analysis in R*, 2024. R package version 3.7-0.

[217] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[218] P. Barrett, *Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor*, in *Conference on the Theory and Application of Cryptographic Techniques*, pp. 311–323, Springer, 1986.

[219] C.-L. Lu, S. Wang, Z. Ji, Y. Wu, L. Xiong, X. Jiang, and L. Ohno-Machado, *WebDISCO: a web service for distributed cox model learning without patient-level data sharing*, *Journal of the American Medical Informatics Association* (2015).

[220] D. Li, W. Lu, D. Shu, S. Toh, and R. Wang, *Distributed cox proportional hazards regression using summary-level information*, *Biostatistics* (2023).

[221] S. Yu, G. Fung, R. Rosales, S. Krishnan, R. B. Rao, C. Dehing-Oberije, and P. Lambin, *Privacy-preserving cox regression for survival analysis*, in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1034–1042, 2008.

[222] Y. Lu, Y. Tian, T. Zhou, S. Zhu, and J. Li, *Multicenter privacy-preserving cox analysis based on homomorphic encryption*, *IEEE Journal of Biomedical and Health Informatics* (2021).

[223] W. Xu, X. Li, Y. Su, B. Wang, and W. Zhao, *Verifiable privacy-preserving cox regression from multi-key fully homomorphic encryption*, *Peer-to-Peer Networking and Applications* (2024).

[224] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, *Learning differentially private recurrent language models*, in *International Conference on Learning Representations*, 2018.

[225] K. Zhu, P. Van Hentenryck, and F. Fioretto, *Bias and variance of post-processing in differential privacy*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11177–11184, 2021.

[226] A. Nitulescu, "zk-snarks: A gentle introduction." `https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf`, 2020.

[227] A. Viand, C. Knabenhans, and A. Hithnawi, *Verifiable fully homomorphic encryption*, *arXiv preprint arXiv:2301.07041* (2023).