Privacy-Preserving Control of Partitioned Energy Resources

Evan Laufer
emlaufer@stanford.edu
Stanford University

Philip Levis
pal@cs.stanford.edu
Stanford University

Ram Rajagopal
ramr@stanford.edu
Stanford University

ABSTRACT

Distributed energy resources are an increasingly important part of the electric grid. We examine the problem of partitioning a distributed energy resource among many users while providing privacy to them. In this model, clients can send requests to a server, the server can verify that the requests are valid and aggregate them, but it cannot see the actual values in the requests. Without privacy, each user is forced to reveal their daily schedule or energy use.

Energy resources add a novel challenge that prior systems do not address: they require verifying limits on private power (a rate over time) and energy (a sum) values. Furthermore, the cryptographic mechanisms must run on embedded energy control systems.

We describe Weft, a novel cryptographic system that verifies both power (rate) and energy (integral) constraints on private client values and aggregates them. The key insight behind the approach is to rely on additively homomorphic secret shares, which allows servers to compute sums from rates. We present 3 cryptographic proof systems with different system trade-off for embedded systems: bit-splitting proofs minimize memory use, sorting proofs minimize computation, and commitment proofs minimize network communication. Using bit-splitting proofs, it takes an IoT client using a CortexM microcontroller 4 minutes of compute time to privately control its share of an energy resource for a day at 20s granularity.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

energy storage, distributed energy resources, privacy

ACM Reference Format:
Evan Laufer, Philip Levis, and Ram Rajagopal. 2024. Privacy-Preserving Control of Partitioned Energy Resources. In *The 15th ACM International Conference on Future and Sustainable Energy Systems (E-Energy '24), June 04–07, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3632775.3661988

1 INTRODUCTION

Distributed energy resources (DERs) are an increasingly important part of the electric grid. There are now over 500,000 Tesla PowerWalls installed in the United States [43], providing 7GW of power, which is more than half of the 13GW [53] provided by utility battery installations. Distributed energy resources differ from their utility counterparts in that they are owned and controlled by consumers and third parties.

Some distributed energy resources are partitioned among many users. For example, a community solar/battery installation may sell partitions to people in a neighborhood or power domain. Sharing a communal resource in this way has many advantages: individuals can join and leave the system, resources can be put in good locations, and it allows people who cannot install resources in their home to use them.

One problem that partitioned energy resources introduce is a lack of privacy. Because each user operates a share of the large resource, the controller of the resource (e.g., the battery storage provider) can see how each person is using it. For example, the provider can observe that a home starts requesting 40kW at 3PM, suggesting they are charging an electric vehicle.

Recent work has considered a similar scenario, where clients jointly control an energy resource [62]. Clients privately aggregate their energy demand, which is sent to a resource operator that minimizes the joint cost of the energy. This paper considers a different scenario, where clients own and control partitions of an energy resource. A major benefit of this approach is that clients can individually optimize their own energy cost. Compared to the joint control setting, this introduces a key challenge: the resource operator must verify that the commands from the client are valid. For example, in the case of a privately shared battery, the battery owner needs to verify that a client isn't requesting more power than their maximum, nor that they are undercharging or overcharging their battery. Section 8 describes how this leads to different cryptographic elements in this work.

Adapting private aggregation to partitioned energy resources has three challenges. First, distributed energy resources require verifying both an instantaneous value (power) and its integral (energy). Second, values are commands to an active device, and the timing of these commands can leak sensitive information about the client's usage of the resource. Third, prior techniques focus on traditional client/server computing systems that have GHz of CPU and GB of RAM. Distributed energy resources, in contrast, often have embedded controllers, with MHz of CPU and kB-MB of RAM.

This paper describes Weft, a secure system which addresses these challenges. First, Weft enforces constraints over power (rates) and energy (integrals) using the key observation that the additive homomorphism in systems like Prio and a local state variable allow servers to compute and constrain secret integrals over time. Second, to provide temporal secrecy, instead of sending commands to the device, clients send schedules of commands at fixed times. Third, Weft can run in diverse deployment scenarios, including on resource constrained embedded devices. Weft can use three types of proofs which each minimize one of compute, memory, or network communication.

E-Energy '24, June 04–07, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0480-2/24/06
https://doi.org/10.1145/3632775.3661988

This paper makes the following contributions:

(1) Design of a novel cryptographic system for privacy-preserving control of partitioned energy resources, which can verify both power (rate) and energy (integral) constraints, using three proof systems with different trade-offs:
   (a) Bit-splitting proofs which minimize memory usage.
   (b) Sorting proofs which minimize CPU usage.
   (c) Commitment proofs, which minimize network communication. These proofs are a general-use, novel extension to Bulletproofs to operate over secret-shared data.
(2) A novel optimization (Disjunctive Schedule Compression) for compressing secret shared time series data, at the cost of leaking a small amount of information.
(3) An implementation that is efficient enough to run on resource constrained embedded systems.

2 BACKGROUND

This section provides background on distributed energy resources and three cryptographic techniques that Weft builds on: additive secret sharing, secure aggregation, and cryptographic proofs.

2.1 Distributed Energy Resources

Distributed energy resources (DERs) are energy resources that are controlled by consumers and third-parties, instead of the energy company. DERs are a growing segment of the grid. For example, Powerwall, Tesla's home battery, is now installed in over 500,000 homes in the United States, with over half installed in the last two years [43]. In total, they can provide 7GW of power to the grid, which is more than half of the 13GW that can be provided by battery installations owned by energy companies [53].

Because DERs are controlled by many parties and distributed across the grid, they often have poor utilization [52]. For example, typically home batteries are only used to minimize energy cost. Recent work in academia and industry has created systems to increase the utilization of DERs through aggregation and sharing [2, 26, 44, 52].

2.2 Additive Secret Sharing

Secret sharing is a technique that splits a value v into n secret *shares*, $[v]_1, ..., [v]_n$.[1] Each share individually reveals nothing about v, and recovering the original value requires having all of the shares [17, 59]. Additive Secret Sharing, used in this paper, operates on values over a modulus, which wrap around on overflow (e.g. standard 64-bit unsigned integers).

In additive secret sharing, $n-1$ shares ($[v]_1 \dots [v]_{n-1}$) are random numbers, and the remaining share ($[v]_n$) is v with every random number subtracted from it ($[v]_n = v - [v]_1 - [v]_2 - \dots - [v]_{n-1}$). To recover v, a user adds together all of the shares: the random numbers cancel and v remains. If a user holds fewer than n shares, they are unable to learn any information about v because the shares appear completely random.

Additive secret sharing is *additively homomorphic*, meaning one can add or subtract values by adding or subtracting their secret shares. For example, suppose there are two values, v, and w, each split into secret shares. Adding up all of the shares of v and w (in any order) will produce $v + w$.

[1] Throughout this paper, $[v]_i$ denotes the i^{th} share of v.

2.3 Secure Aggregation and Prio

Secure aggregation allows a server to compute an aggregate of many values without learning the individual values. It also supports computations in which only some input values are valid. Using secure aggregation, a battery-solar system can receive secret discharge requests from users, check that each request is valid, and compute the total amount of power it should provide to the grid without learning any individual user's request.

A secure aggregation scheme has two security properties.

(1) Client Privacy: A (potentially malicious) server should learn no information about an individual client's values, except for the final sum and that each value is valid.
(2) Server Robustness: If a client sends the server a malicious value, a server can detect and reject it.

Prio is a secure aggregation system [27]. To provide client privacy, Prio adds a non-colluding third party that participates in the protocol. Clients send arithmetic secret shares of their values to each server. By the privacy property of these shares, as long as one server is not malicious, they reveal no information about the underlying value. The servers can individually sum up the client shares into partial sums. They add together their partial sums to reveal the sum of all the client values. To provide server robustness, Prio introduces SNIPs, a type of zero-knowledge proofs [37].

2.4 Bulletproofs

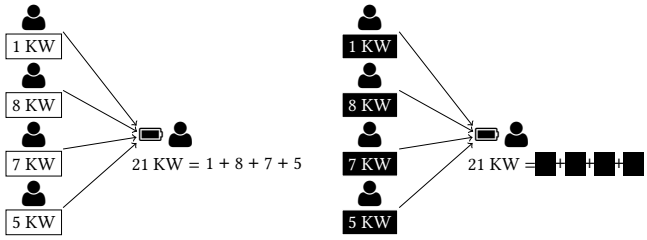
Bulletproofs is an efficient, aggregatable zero-knowledge proof system [22]. Compared to Prio's SNIPs, Bulletproofs require more memory and computation but are much more concise, so require $\approx 10x$ less network communication. For DERs on low-speed long-range wireless links, this can be a necessary trade-off. A Bulletproofs prover publishes a public *commitment* C to w, which binds them to the value of w without revealing any information about w. Later, if the prover reveals w, other parties may verify that w was indeed the committed value. Bulletproofs range proofs prove that the commitment C both commits to w and that $0 \leq w < 2^n$, without revealing w.

3 PROBLEM STATEMENT

This section formulates privacy-preserving control of partitioned energy resources as a security problem, shown in Figure 1.

A client wants to use a partitioned energy resource. Many clients own partitions of the resource, but it is operated by a central server. The server maintains a (possibly negative, in case of charging and discharging) minimum rate, maximum rate, and a maximum amount of the resource that the client can use. The client wants their usage to remain private; the server should learn the sum of all clients' usages but not the usage of any individual client. The server wants to ensure that no client uses a rate outside their allowed bounds, more than the maximum amount, or drains their resource below zero.

To be practical for real-world deployment, the system (and particularly the client) must be efficient. Energy systems often run on low-power embedded devices with limited compute, memory,



(a) Without privacy **(b) With privacy**

Figure 1: Partitioned resource control with and without privacy. Without privacy, the server sees each client's value and adds them together. With privacy, each client's value is encrypted such that the server can sum them without seeing the value.

or bandwidth (e.g., smart meters, energy system controllers). The system assumes the server is commodity hardware, with resources at least equivalent to a recent desktop.

In summary, a system for privacy-preserving control of partitioned energy resources has the following goals:

(1) Client privacy: the resource operator should only learn the sum of all client requests.
(2) Server integrity: the resource operator should be able to detect and reject invalid client requests.
(3) The system should be computationally efficient, able to run on desktop-class systems and in some configurations, clients can be embedded devices.

4 SYSTEM DESIGN

Weft allows private control of a partitioned energy resource. In this model, each client controls a partition of the energy resource, and sends private commands to the resource operator, such as charge or discharge. The resource operator aggregates the client requests. For example, following the approach used in prior work on battery virtualization [52], to calculate whether a partitioned battery should charge or discharge it sums the charge and discharge requests.

The exact commands are private. The resource operator does not know what individual clients do. The resource operator, however, can verify that each command is valid in terms of power and energy: no client can charge or discharge faster than what their partition allows, nor can it overcharge above its partition capacity or discharge below zero.

The rest of this section formally describes this security model, how clients specify their commands in terms of schedules, and how Weft extends secure aggregation to verify the validity of both power and energy.

4.1 Security Model

Weft follows the same security model as Prio [27]. The resource operator has a server, responsible for controlling the energy resource and sending commands to it. There is at least one additional server, operated by a third party (e.g., the ISRG [4]). Weft assumes the third party does not collude with the resource operator. This is a practical assumption because organizations such as the Internet Security Research Group provide this as a service. For example, in a deployment of secure aggregation used for COVID-19 exposure notifications, the servers were controlled by the NIH and ISRG [4].

The clients know their own requests and the state of their partition of the energy resource, but they should not learn any information about other clients. The servers should learn whether each request is valid and the sum of the power across all client's requests, but no other client information. Weft assumes that the clients and servers communicate over authenticated, encrypted channels.

Servers are trusted for request *integrity*, meaning they will not modify client requests. However, servers are untrusted for request *privacy*, meaning they may try to learn information about client requests. This information may be from the request itself, by relating different requests, from the timing of requests, etc. Clients are trusted for neither *integrity* nor *privacy*; they may attempt to learn other client's information or submit malformed requests.

4.2 Data Model

Like prior work on secure aggregation in the non-colluding third-party model, Weft uses secret sharing to preserve client privacy [27]. A client submits a power request by splitting the request into secret shares and sending one share to each server. Using the additive homomorphism of the sharing scheme, the servers aggregate the shares of different clients, hiding the client's private data in the sum. The resource operator recombines the aggregated shares to learn the true aggregate.

Standard RESTful APIs for energy resources allow clients to control a resource by setting its charge/discharge value. This value replaces the prior one. While this can be achieved with secret shares (simply cache values and reaggregate on a new value), it leaks information and therefore violates privacy. Whenever a request comes in from a client, the resource operator sees how the aggregate changes and can compute how the client's request changed.

To avoid this information leak, Weft encodes requests as *schedules*, an array of power values, similarly to prior work [62]. Each element in the schedule specifies the value for a time interval. The energy resource defines this interval and the length of a schedule. Shortly before the start of each epoch, a client sends its schedule for that epoch. Schedules do not leak information because they decouple changes from the timing of messages.

To ensure the client request is valid, the servers must check two things: a *rate constraint* and an *integral constraint*. More formally, for some minimum (possibly negative) rate n, some maximum rate m, and a maximum usage e, these constraints over a schedule S are

(1) Rate Constraint: $n \leq S[i] \leq m$ for all $0 \leq i < |S|$.
(2) Integral Constraint: $0 \leq \sum_{i=0}^{j} S[i] \leq e$ for all $0 \leq j < |S|$

The minimum rate, maximum rate, and maximum usage may all be different values for each client, and we assume that they are known to the servers and the resource operator.

To ensure that client requests are valid, the system uses zero-knowledge proofs over secret shared data. The client constructs zero-knowledge proofs which prove that their schedule S satisfies both the rate and integral constraints. They send the proofs to the servers, who jointly verify them over their secret shares. If the proofs are valid, the servers are convinced that S satisfies both constraints. If any proof is invalid, the servers reject S.

4.3 Checking Integral Constraints

Both the rate and integral constraints require checking *range predicates* (a value is within a range). To prove the rate constraint, the client generates a zero-knowledge proof for each value $S[i]$ in its schedule, which proves $n < S[i] < m$. We defer a description of how these proofs are generated to Section 5.

While the rate constraint is a straight-forward application of range predicate proofs, proving the integral constraint requires extra steps. The key insight behind checking integral constraints is that because secret shares are additively homomorphic, one can compute the integral by keeping a running sum of the rates.

Recall the integral constraint requires proving that for some maximum usage e, over a schedule S

$$0 \leq \sum_{i=0}^{j} S[i] \leq e \text{ for all } 0 \leq j < |S|$$

Because the schedules are arrays, the integral over $[0, t]$ is the partial sum of the values from 0 to t. The servers can compute secret shares of the integral at each time-slice by summing up the secret shares of the schedule.

Once the servers have secret shares of the integral at each time-slice, the client can use the same range proofs used to prove the rate constraint for the integral constraint. Instead of proving that $n \leq S[i] \leq m$, the client computes the integral and proves that $0 \leq \sum_{i=0}^{t} S[i] \leq e$. The servers can verify these proofs using their secret shares of the integrals.

Integrals across schedules. In the partitioned battery application, the amount of energy remaining in the client's partition must never be below 0 or above the maximum possible energy e. This constraint must hold across multiple schedules: the value at the start of the schedule depends on the prior one. The amount of energy remaining in the client's partition should also remain private, because it reveals information about the client's energy usage.

Weft extends the integral constraint across schedules. Each server keeps a secret share representing the current sum up to this point (denoted $[e_c]$). Weft modifies the integral constraint to

$$0 \leq e_c + \sum_{i=0}^{n} S[i] \leq e \text{ for all } 0 \leq n < |S|$$

The servers compute shares of each partial sum using the additive homomorphism of the secret shares. The client generates a range proof for each partial sum, which convinces the servers their schedule satisfies the constraint.

Once the current schedule is finished, the servers update their shares of e_c using the shares of the schedule

$$[e_c] = [e_c] + \sum_{i=0}^{|S|} [S[i]]$$

Again, the servers are able to update $[e_c]$ locally with no help from the client or other servers because the secret sharing scheme is additively homomorphic.

5 RANGE PROOFS

The client needs to prove to the server that their secret value v is within some interval $[n, m]$. This section gives three types of proofs for these predicates. Full descriptions of Weft's range proofs are given in Appendix A, and security proofs are given in Appendix B.

Proof	Compute (s)	Memory (kB)	Communication (kB)
Bit-splitting	0.13	105	723
Sorting	0.04	638	343
Commitment	34.71	262×10^3	24

Table 1: The amount of compute, memory, and communication needed to prove a schedule with 5760 values (one value sampled every 15 seconds over one day).

The first proof is a bit-splitting argument implemented using Prio. Informally, if each value $S[i]$ is expressible within n bits, then $S[i]$ must be within the interval $[0, 2^n)$. The proof can extend to arbitrary intervals using additive homomorphism.

The second proof is based on sorting, and is also implemented using Prio. The client convinces the servers that the values in a schedule S are within an interval $[n, m]$ by providing a sorted version of the schedule $\hat{S}$. If the first and last values in $\hat{S}$ are within $[n, m]$, then the entire schedule S must also be within $[n, m]$.

The third proof, which we call commitment proofs, is based on Bulletproofs, but adds novel modifications to allow it to be used over secret shared data, which is of independent interest. The key insight is to have the servers construct a commitment to the original, unshared values using the additive homomorphism of the commitment scheme.

Table 1 summarizes trade-offs to prove the validity of 5760 values with each proof system. The range proofs are the most expensive part of the client and server, and ideally there would be one proof that is efficient in compute, memory, and communication. Unfortunately, techniques which reduce the cost of one resource increase the cost of others. For example, the bit-splitting and sorting proofs are very compute efficient. However, they require a lot of network communication. On the other hand, the commitment proofs are very communication efficient, but require large amounts of compute and memory.

5.1 Disjunctive Schedule Compression

For large schedules, range proofs use too much RAM to run on embedded devices. One way to reduce this cost is to increase the granularity of the schedule. However, this means clients no longer have as much flexibility with when they can change their values.

To reduce the schedule sizes while still giving clients flexibility, the system may implement an optimization that allows clients to only send values at times when some client changes their value. This reduces the cost for applications where clients are often sending duplicate values within their schedule, while providing the full flexibility of a larger schedule. However, it requires leaking when some client changes their schedule.

Weft implements the optimization as an additional round before clients send their schedules and proofs. During this round, the clients send the servers secret shares of bits which encode whether or not their value changes during that time slice. The servers compute the multiplication of the clients bits, and reveal the result to all the clients. The clients only send values and proofs for the bits which indicate that some client intends to change their value.

More precisely, at the beginning of the first round, each client computes $x[i] = (S[i] == S[i-1])$ for all $1 \leq i \leq |S|$ and sends a boolean secret share of $x[i]$ to each server. $x[i]$ is 1 if the client didn't change the value in their schedule between time slice $i-1$ and i, and 0 otherwise. The servers maintain an accumulator vector acc, where $acc[t]$ is the value of the accumulator for time t. This accumulator is initialized to shares of all 1s (the initial state does not need to be secret, so the servers can initialize with no communication). On a set of bits x the servers compute $acc = acc \circ x$ where $\circ$ is an entrywise product. The entrywise product can be computed using standard MPC techniques [11, 68]. Once an x is received from all clients, the servers reveal the values of acc to each other and the clients. Then, for time t, the clients only send values and proofs when $acc[t] = 1$

6 IMPLEMENTATION

Weft is implemented as a Linux process and as a TockOS process running on the nRF52840 microcontroller. The system uses using modified versions of the `libprio` crate [3] and the `bulletproofs` [29] crate to implement the range proofs. For the Tock implementation, we modified `libprio` to materialize less data in memory to support larger schedule sizes on RAM constrained embedded systems. Most of these modifications were straightforward.

7 EVALUATION

This section evaluates Weft and whether it is efficient enough to be deployed and used in distributed energy resources. It does so by examining three questions:

- What are the CPU, memory, and communication costs of Weft on clients, which may be IoT systems?
- What are the CPU costs of the Weft on servers? (The memory and communication costs are tiny for server-class systems.)
- What savings does disjunctive schedule compression provide on realistic DER control traces?

7.1 Methodology

We measure client costs on a desktop-class machine as well as a microcontroller-based embedded system. We measure server costs on only the desktop. The embedded system has an nRF52840 microcontroller, a system-on-a-chip (SoC) integrating a 64 MHz 32-bit Cortex-M4 microcontroller with 1MB of flash for code and 256kB of RAM [1]. The desktop has an Intel i9-129000KF processor with 8 performance and 8 efficiency cores. Each performance core can run at up to 5.2GHz. The system has 30MB of shared L3 and 14MB total L2 cache. Main memory is DDR4-3200.

The primary variable we manipulate is the schedule size, in terms of number of values.

7.2 RAM use

Figure 2 shows the peak client heap of Weft using the bit-splitting and sorting proofs. To prove a schedule with 10,000 elements, the sorting strategy uses a peak of 2.4MB of RAM, the bit-splitting strategy uses 191kB. The commitment proofs require 524MB of RAM for the same schedule, which exceeds the memory of most IoT devices. Bit-splitting is the most RAM-efficient approach, using 8% the RAM of sorting and 0.04% the RAM of commitment proofs.

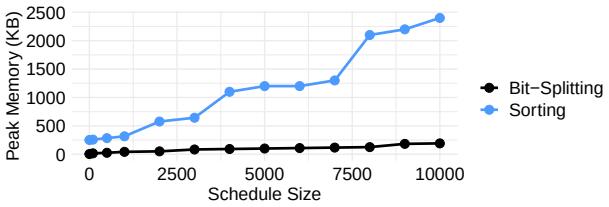


Figure 2: Peak memory usage of the client using the Bit-Splitting and Sorting proofs.

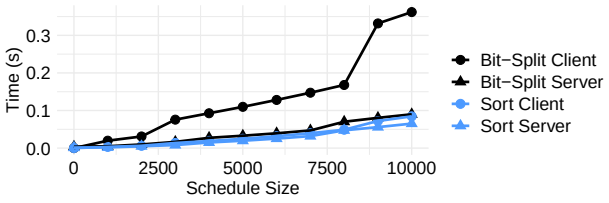


Figure 3: Compute time by the client and server when using the bit-splitting or sorting proofs.

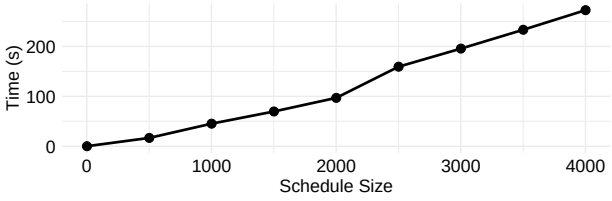


Figure 4: Time spent by the client generating proofs for various schedule sizes on an nRF52840 microcontroller. For a schedule of size 4000, the client takes 4.5 minutes.

The 191kB required for bit-splitting approaches the RAM available on the nRF52840. In practice, its 256kB has to be shared between applications, the kernel, and networking stack. In our execution environment, an application is limited to 96kB of RAM. This is sufficient to prove a 4,000 element schedule, so a day-long schedule with an interval of 22 seconds. A 1,440 element schedule uses 45kB.

7.3 CPU Performance

Figure 3 shows the time it takes the desktop to compute the client and server portion of bit-splitting and sorting proofs. It can prove the validity of a 10,000 element schedule with the bit-splitting strategy in 0.36 seconds. Using the sorting strategy, it can prove the same schedule in 0.09 seconds. Server verification of the two Prio-based proof strategies takes 0.085 and 0.065 seconds respectively. Commitment proofs is the most computationally intensive approach. A client can prove the validity of a schedule of 10,000 elements in 73.59 seconds. It takes a server 9.99 seconds to verify the proofs.

Figure 4 shows the compute time for the bit-splitting client when running as a Tock [45] process on the nRF52840 microcontroller. A

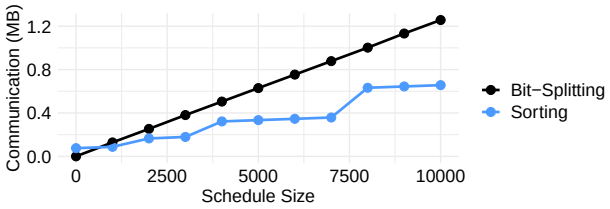


Figure 5: Amount of data sent from the client to the leader server when using the bit-splitting or sorting proofs.

# steps	Compressed size	% reduction
-	1440	0%
10000	1158	20%
1000	697	52%
100	215	85%
10	28	98%

Table 2: The table shows the compressed size of a single-day battery discharge schedule when discretized to various numbers of steps. The first row shows the uncompressed size (1440).

schedule of 1,440 elements takes 66s to compute, while a maximum-sized schedule of 4,000 elements takes 4.5 minutes; as the schedule is for a day, this is an acceptable cost.

7.4 Communication

Figure 5 shows the number of bytes the bit-splitting and sorting proofs communicate between the clients and servers. We assume server-to-server communication is not a bottleneck (the number of bytes is tiny for modern Internet speeds).

The sorting strategy requires 0.66 MB for a 10,000 element schedule and 159kB for a 1,440 element schedule. The bit-splitting strategy, which can execute on an embedded microcontroller, requires the most communication. It sends 1.13MB for a 10,000 element schedule and 183kB for 1,440 elements.

Commitment proofs are the most efficient strategy, sending 41.5 KB for a schedule of 10,000 values. This is close to optimal – the values in the schedule themselves require 40 KB of data. For a 1,440 element schedule, commitment proofs send 7.1kB.

7.5 Disjunctive Schedule Compression

Disjunctive schedule compression has negligible RAM and CPU overhead: the client only needs to compute the secret shares of the schedule changes, and the servers only performs bitwise operations. For communication, the clients must send bits which represent when their schedule changes, and the servers must perform boolean MPC to multiply all the clients changes together.

To evaluate the effectiveness of schedule compression, we use a trace of a battery-solar system on a home in California. Because these values are measurements of observed output from a control system, rather than the commands issued by the control system, they are noisy. We therefore discretize the power values into bins.

We discretize the power values by rounding them down to the nearest value. Table 2 shows the results, showing the size of the compressed schedule when power is discretized into 10, 100, 1000, or 10000 values, representing steps of 1kW to 1W. With high resolution (e.g. $n = 10000$), compression reduces the schedule size by 282 values. At a resolution of $n = 1000$ (10W steps), compression cut the schedule size by half. At 1kW granularity the schedule is cut to 28 items. This represents a *lower bound* on the size of the schedule after optimization – in reality it compresses to the union of the changes among all clients.

8 RELATED WORK

Wang, Chau, and Zhou propose using multi-party computation (MPC), zero-knowledge proofs, and Blockchains to reduce costs among users who share an energy storage unit while preserving user privacy [62]. This work differs from Weft in two key ways. First, in its model, many users have a *s*hared resource and the system minimizes the total cost across all the users. In Weft's model, many users have *p*artitions of an energy resource, which they individually control, and the system hides their commands and use from the owner of the storage unit. Second, this prior work is intended to run on desktop-class computing systems.

There is a deep literature on private collection of smart meter data for billing, demand response, and aggregate statistics [5, 7, 9, 21, 23, 24, 32–34, 36, 38, 42, 46–50, 56, 57, 60, 64]. These approaches focus on private aggregation in the single-server setting, as opposed private resource control to the non-colluding third-party model that Weft uses; they solve a different problem and must rely on more expensive cryptographic mechanisms.

There is a long line of research into secure aggregation using a single server (not requiring a non-colluding third party) [16, 31, 39, 60]. While these techniques could be readily applied to power values, extending their schemes to support proofs over integrals (energy) is an open problem.

Another key area of related work is on secure aggregation in the non-colluding server model outside the energy application domain. Prio+ [6] builds on Prio by reducing client to server communication cost by supporting boolean secret shares, at the cost of higher server compute and communication.

There is a long line of research into efficient zero-knowledge proof systems, and those used within secure aggregation schemes are only a small portion of it [12–15, 18, 22, 25, 28, 35, 41, 51, 55, 58, 61, 65–67, 69]. The commitment proofs used in Weft make a contribution to the zero-knowledge literature by extending Bulletproofs for use over secret shares.

9 CONCLUSION

This paper presents Weft, a system for privacy preserving control of partitioned energy resources. The system is efficient, and can be configured to run on resource constrained embedded systems.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Electricity under Award Number DE-OE0000919 and NSF grant #2303639.

REFERENCES

[1] 2021. nRF52840 Product Specification v1.7. https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.7.pdf.

[2] 2024. Join the Tesla Virtual Power Plant. https://www.tesla.com/support/energy/tesla-virtual-power-plant-pge.

[3] 2024. libprio-rs crate. https://github.com/divviup/libprio-rs.

[4] Josh Aas and Tim Geoghegan. 2020. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/.

[5] Gergely Ács and Claude Castelluccia. 2011. I Have a DREAM! (DiffeRentially privatE smArt Metering). In *Information Hiding*, Tomáš Filler, Tomáš Pevný, Scott Craver, and Andrew Ker (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 118–132.

[6] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. 2021. Prio+: Privacy Preserving Aggregate Statistics via Boolean Shares. Cryptology ePrint Archive, Report 2021/576. https://eprint.iacr.org/2021/576.

[7] Khalid Alharbi and Xiaodong Lin. 2012. LPDA: A lightweight privacy-preserving data aggregation scheme for smart grid. In *2012 International Conference on Wireless Communications and Signal Processing (WCSP)*. 1–6. https://doi.org/10.1109/WCSP.2012.6542936

[8] Apple and Google. 2021. *Exposure Notification Privacy-preserving Analytics (ENPA) White Paper*. Technical Report. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf

[9] Mungyu Bae, Kangho Kim, and Hwangnam Kim. 2016. Preserving privacy and efficiency in data communication and aggregation for AMI network. *Journal of Network and Computer Applications* 59 (2016), 333–344. https://doi.org/10.1016/j.jnca.2015.07.005

[10] Stephanie Bayer and Jens Groth. 2012. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *Advances in Cryptology – EUROCRYPT 2012*, David Pointcheval and Thomas Johansson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 263–280.

[11] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–432.

[12] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2019. Scalable Zero Knowledge with No Trusted Setup. 701–732. https://doi.org/10.1007/978-3-030-26954-8_23

[13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. Cryptology ePrint Archive, Report 2013/507. https://eprint.iacr.org/2013/507.

[14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2013. Succinct Non-Interactive Arguments for a von Neumann Architecture. Cryptology ePrint Archive, Report 2013/879. https://eprint.iacr.org/2013/879.

[15] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. 276–294. https://doi.org/10.1007/978-3-662-44381-1_16

[16] Fabrice Benhamouda, Marc JOYE, and Benoît Libert. 2015. A New Framework for Privacy-Preserving Aggregation of Time-Series Data. *ACM Transactions on Information and System Security* 18 (07 2015). https://doi.org/10.1145/2873069

[17] G. R. Blakley. 1979. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society, Los Alamitos, CA, USA, 313. https://doi.org/10.1109/AFIPS.1979.98

[18] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. 67–97. https://doi.org/10.1007/978-3-030-26954-8_3

[19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs. Cryptology ePrint Archive, Paper 2019/188. https://eprint.iacr.org/2019/188 https://eprint.iacr.org/2019/188.

[20] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. 2018. Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. Cryptology ePrint Archive, Report 2018/380. https://eprint.iacr.org/2018/380.

[21] Fábio Borges and Max Mühlhäuser. 2014. EPPP4SMS: Efficient Privacy-Preserving Protocol for Smart Metering Systems and Its Simulation Using Real-World Data. *IEEE Transactions on Smart Grid* 5, 6 (2014), 2701–2708. https://doi.org/10.1109/TSG.2014.2336265

[22] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. 315–334. https://doi.org/10.1109/SP.2018.00020

[23] Le Chen, Rongxing Lu, Zhenfu Cao, Khalid AlHarbi, and Xiaodong Lin. 2015. MuDA: Multifunctional data aggregation in privacy-preserving smart grid communications. *Peer-to-Peer Networking and Applications* 8, 5 (01 Sep 2015), 777–792. https://doi.org/10.1007/s12083-014-0292-0

[24] Xin Chen, Xiaolei Dong, Zhenfu Cao, Jiachen Shen, Yuanjian Zhou, and Jiawei Qian. 2021. SSDA: A Privacy-Preserving and Fault-Tolerant Data Aggregation Scheme Based on Secret Sharing in Smart Grids. In *International Conference on Big Data and Social Sciences*. https://doi.org/10.1109/ICBDSS53610.2021.00035

[25] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. 2019. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. Cryptology ePrint Archive, Report 2019/1047. https://eprint.iacr.org/2019/1047.

[26] Tzi cker Chiueh, Mao-Cheng Huang, Kai-Cheung Juang, Shih-Hao Liang, and Welkin Ling. 2018. Virtualizing Energy Storage Management Using RAIBA. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 187–198. https://www.usenix.org/conference/atc18/presentation/chiueh

[27] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. *CoRR* abs/1703.06255 (2017). arXiv:1703.06255 http://arxiv.org/abs/1703.06255

[28] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2014. Geppetto: Versatile Verifiable Computation. Cryptology ePrint Archive, Report 2014/976. https://eprint.iacr.org/2014/976.

[29] Dalek Cryptography. 2024. Bulletproofs crate. https://github.com/dalek-cryptography/bulletproofs.

[30] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. 2023. Verifiable Distributed Aggregation Functions. Cryptology ePrint Archive, Report 2023/130. https://eprint.iacr.org/2023/130.

[31] Keita Emura. 2017. Privacy-Preserving Aggregation of Time-Series Data with Public Verifiability from Simple Assumptions. 193–213.

[32] Chun-I Fan, Shi-Yuan Huang, and Yih-Loong Lai. 2014. Privacy-Enhanced Data Aggregation Scheme Against Internal Attackers in Smart Grid. *IEEE Transactions on Industrial Informatics* 10, 1 (2014), 666–675. https://doi.org/10.1109/TII.2013.2277938

[33] Hongbin Fan and Changbing Huang. 2021. Blockchain-based data aggregation scheme for fault-tolerant privacy-preserving in smart grid. In *2021 8th International Forum on Electrical Engineering and Automation (IFEEA)*. 376–380. https://doi.org/10.1109/IFEEA54171.2021.00169

[34] Mochan Fan and Xiaohong Zhang. 2019. Consortium Blockchain Based Data Aggregation and Regulation Mechanism for Smart Grid. *IEEE Access* 7 (2019), 35929–35940. https://doi.org/10.1109/ACCESS.2019.2905298

[35] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. https://eprint.iacr.org/2019/953.

[36] Flavio D. Garcia and Bart Jacobs. 2011. Privacy-Friendly Energy-Metering via Homomorphic Encryption. In *Security and Trust Management*, Jorge Cuellar, Javier Lopez, Gilles Barthe, and Alexander Pretschner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 226–238.

[37] S Goldwasser, S Micali, and C Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA) *(STOC '85)*. Association for Computing Machinery, New York, NY, USA, 291–304. https://doi.org/10.1145/22145.22178

[38] Zhitao Guan and Guanlin Si. 2017. Achieving privacy-preserving big data aggregation with fault tolerance in smart grid. *Digital Communications and Networks* 3, 4 (2017), 242–249. https://doi.org/10.1016/j.dcan.2017.08.005 Big Data Security and Privacy.

[39] Marc Joye and Benoît Libert. 2013. A Scalable Scheme for Privacy-Preserving Aggregation of Time-Series Data. 111–125. https://doi.org/10.1007/978-3-642-39884-1_10

[40] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. 2020. MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs. Cryptology ePrint Archive, Report 2020/278. https://eprint.iacr.org/2020/278.

[41] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. 2020. MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs. 2129–2146.

[42] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. 2011. Privacy-Friendly Aggregation for the Smart-Grid. In *Privacy Enhancing Technologies*, Simone Fischer-Hübner and Nicholas Hopper (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 175–191.

[43] Fred Lambert. 2023. *Tesla has now installed over 500,000 Powerwalls*. https://electrek.co/2023/06/16/tesla-installed-over-500000-powerwalls/

[44] Stephen Lee, Prashant Shenoy, Krithi Ramamritham, and David Irwin. 2021. AutoShare: Virtual community solar and storage for energy sharing. *Energy Informatics* 4, 1 (12 Jul 2021), 10. https://doi.org/10.1186/s42162-021-00144-w

[45] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. 2017. Multiprogramming a 64kB Computer Safely and Efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 234–251. https://doi.org/10.1145/3132747.3132786

[46] Fengjun Li, Bo Luo, and Peng Liu. 2010. Secure Information Aggregation for Smart Grids Using Homomorphic Encryption. In *2010 First IEEE International Conference on Smart Grid Communications*. 327–332. https://doi.org/10.1109/SMARTGRID.2010.5622064

[47] Hongwei Li, Xiaodong Lin, Haomiao Yang, Xiaohui Liang, Rongxing Lu, and Xuemin Shen. 2014. EPPDR: An Efficient Privacy-Preserving Demand Response Scheme with Adaptive Key Evolution in Smart Grid. *IEEE Transactions on Parallel and Distributed Systems* 25, 8 (2014), 2053–2064. https://doi.org/10.1109/TPDS.2013.124

[48] Xiong Li, Shanpeng Liu, Fan Wu, Saru Kumari, and Joel J. P. C. Rodrigues. 2019. Privacy Preserving Data Aggregation Scheme for Mobile Edge Computing Assisted IoT Applications. *IEEE Internet of Things Journal* 6, 3 (2019), 4755–4763. https://doi.org/10.1109/JIOT.2018.2874473

[49] Rongxing Lu, Xiaohui Liang, Xu Li, Xiaodong Lin, and Xuemin Shen. 2012. EPPA: An Efficient and Privacy-Preserving Aggregation Scheme for Secure Smart Grid Communications. *IEEE Transactions on Parallel and Distributed Systems* 23, 9 (2012), 1621–1631. https://doi.org/10.1109/TPDS.2012.86

[50] Lingjuan Lyu, Karthik Nandakumar, Ben Rubinstein, Jiong Jin, Justin Bedo, and Marimuthu Palaniswami. 2018. PPFA: Privacy Preserving Fog-Enabled Aggregation in Smart Grid. *IEEE Transactions on Industrial Informatics* 14, 8 (2018), 3733–3744. https://doi.org/10.1109/TII.2018.2803782

[51] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings. Cryptology ePrint Archive, Report 2019/099. https://eprint.iacr.org/2019/099.

[52] Sonia Martin, Nicholas Mosier, Obi Nnorom, Yancheng Ou, Liana Patel, Oskar Triebe, Gustavo Cezar, Philip Levis, and Ram Rajagopal. 2022. Software Defined Grid Energy Storage. In *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (Boston, Massachusetts) *(BuildSys '22)*. Association for Computing Machinery, New York, NY, USA, 218–227. https://doi.org/10.1145/3563357.3564082

[53] Kassia Micek and Justine Coyne. 2023. US battery storage: Capacity tops 12.5 GW in Q2; 3.5 GW planned in Q3. https://www.spglobal.com/commodityinsights/en/market-insights/latest-news/electric-power/082523-us-battery-storage-capacity-tops-125-gw-in-q2-35-gw-planned-in-q3

[54] C. Andrew Neff. 2001. A Verifiable Secret Shuffle and Its Application to e-Voting. 116–125. https://doi.org/10.1145/501983.502000

[55] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. 238–252. https://doi.org/10.1109/SP.2013.47

[56] Alfredo Rial and George Danezis. 2011. Privacy-Preserving Smart Metering. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society* (Chicago, Illinois, USA) *(WPES '11)*. Association for Computing Machinery, New York, NY, USA, 49–60. https://doi.org/10.1145/2046556.2046564

[57] Sushmita Ruj and Amiya Nayak. 2013. A Decentralized Security Framework for Data Aggregation and Access Control in Smart Grids. *IEEE Transactions on Smart Grid* 4, 1 (2013), 196–205. https://doi.org/10.1109/TSG.2012.2224389

[58] Srinath Setty. 2019. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2019/550. https://eprint.iacr.org/2019/550.

[59] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (nov 1979), 612–613. https://doi.org/10.1145/359168.359176

[60] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data.

[61] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. 2015. Efficient RAM and control flow in verifiable outsourced computation.

[62] Nan Wang, Sid Chi-Kin Chau, and Yue Zhou. 2021. Privacy-Preserving Energy Storage Sharing with Blockchain and Secure Multi-Party Computation. *CoRR* abs/2111.02005 (2021). arXiv:2111.02005 https://arxiv.org/abs/2111.02005

[63] Nan Wang and Sid Chi-Kin Chau. 2022. Flashproofs: Efficient Zero-Knowledge Arguments of Range and Polynomial Evaluation with Transparent Setup. In *Advances in Cryptology – ASIACRYPT 2022*, Shweta Agrawal and Dongdai Lin (Eds.). Springer Nature Switzerland, Cham, 219–248.

[64] Xiaodi Wang, Yining Liu, and Kim-Kwang Raymond Choo. 2021. Fault-Tolerant Multisubset Aggregation Scheme for Smart Grid. *IEEE Transactions on Industrial Informatics* 17, 6 (2021), 4065–4072. https://doi.org/10.1109/TII.2020.3014401

[65] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2020. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. Cryptology ePrint Archive, Report 2020/925. https://eprint.iacr.org/2020/925.

[66] Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2022. Orion: Zero Knowledge Proof with Linear Prover Time. 299–328. https://doi.org/10.1007/978-3-031-15985-5_11

[67] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field. Cryptology ePrint Archive, Report 2021/076. https://eprint.iacr.org/2021/076.

[68] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. 162–167. https://doi.org/10.1109/SFCS.1986.25

[69] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2018. vRAM: Faster Verifiable RAM with Program-Independent Preprocessing. 908–925. https://doi.org/10.1109/SP.2018.00013

A FULL RANGE PROOFS

A.1 Bit-splitting Proofs using Prio

The first proof is based on bit-splitting, a technique that has been used extensively in prior zero-knowledge work [22, 63].

The client has a schedule S, which they want to prove only contains values in an interval $[n, m]$. Because this proof strategy uses Prio, the values in the schedule S (and secret shares) must be values in a finite field $\mathbb{F}$, which is much larger than $[n, m]$.

To prove that a value $s \in S$ is in $[n, m]$, the client proves that $s - n \in [0, 2^b)$ and $m - s \in [0, 2^b)$, where $b = \lceil \log_2(m - n) \rceil$. This follows from arithmetic over the inequality $n \leq s \leq m$.

Let v be either $s - n$ or $m - s$. To prove that $v \in [0, 2^b)$, the client shows that v fits in b bits by sending the servers secret shares of each bit of v and prove that the bits are valid. Let v_i denote the i^{th} bit of v. A valid bit-decomposition of v in b bits has the following properties:

(1) Summation: $v = 2^0 v_0 + 2^1 v_1 + ... + 2^{b-1} v_{b-1}$
(2) Well-Formedness: $v_i \in \{0, 1\}$ for all $i \in 0...b - 1$

The servers check both properties.

The summation condition can be checked using the additive homomorphism of the secret sharing scheme. Each server computes $v - 2^0 v_0 + 2^1 v_1 + ... + 2^{n-1} v_{n-1}$ over their local shares, and they reveal the result to the other servers. If the result is 0, then $v = 2^0 v_0 + 2^1 v_1 + ... + 2^{n-1} v_{n-1}$ as required.

The second condition cannot be checked using the additive homomorphism because it is nonlinear. The client generates a Prio proof that each v_i is 0 or 1, using the validity predicate $p_b(x) = x(x - 1)$.

Batching Prio Proofs. The cost for the Prio bit-checking proofs is large – over an entire schedule S, the size of the proofs scales linearly with $|S|$. To reduce this cost, Weft uses the G-gate optimization [19] which batches repeated proofs into a single proof of size $O(\sqrt{|S|})$. Applying this optimization to repeated bit-checking predicates in Prio has been used in previously deployed systems [8].

Efficiency. The bit-splitting proof is the most memory efficient proof. Because the predicates being checked by Prio are degree-2, the memory footprint of the FFTs used is small. Batching the proofs does increase the memory usage to $O(\sqrt{|S|})$, but this is still efficient enough to run within 256 KB of memory. However, the bit-splitting proof is not communication efficient, because the client must send a secret share of each bit, which amounts to $|S| \cdot b$ extra field elements. This is 640kB with $|S| = 10,000$, $b = 16$, and a 32-bit field.

A.2 Sorting Proofs using Prio

The second technique uses sorting, and is based on is based on techniques used for permutation and RAM checking in ZK-SNARKS [20, 40, 54]. The client sends the servers a sorted list $\hat{S}$ which contains the schedule S, n, and m. The servers use Prio proofs to check that $\hat{S}$ is sorted and contains all values in S. They reveal

the first and last value in the sorted list, which they confirm are equal to n and m respectively. If the client attempts to provide a value which is outside of $[n, m]$, then either $\hat{S}$ is improperly sorted, or the value will be revealed by the servers.

To catch a lying client, the servers check:

(1) $\hat{S}[0] = n$ and $\hat{S}[|S| - 1] = m$
(2) $\hat{S}$ is sorted
(3) $\hat{S}$ contains all values in S

The first condition can be checked by revealing $\hat{S}[0] = n$ and $\hat{S}[|S| - 1] = m$. The second and third conditions are more complex to check efficiently.

Ensuring that $\hat{S}$ is sorted requires checking that $\hat{S}[i-1] < \hat{S}[i]$ for all $i \in 1...|S|+2$. One way to check this is using bit-splitting, but this leads to a communication cost blowup because secret shares of all the bits must be sent to the server. To avoid bit-splitting, the clients also add every value in $[n, m]$ to $\hat{S}$. In particular, the client constructs $\hat{S} = Sort(S||[n, n+1, ..., m])$, where $||$ means list concatenation. If the client is honest, and $\hat{S}$ is sorted, then each consecutive value in $\hat{S}$ is at most 1 apart. This is equivalent to checking that $\hat{S}[i] - \hat{S}[i-1] \in \{0, 1\}$ for all $i \in 0...|\hat{S}|$, which the servers check using Prio proofs.

To check that $\hat{S}$ contains all values in S, the servers use a permutation argument from Bayer et al [10]. The core idea is to encode $S||[n, ..., m]$ and $\hat{S}$ as the following polynomial:

$$H(r) = \prod_{s \in (S||[n,...,m])} (r - s) - \prod_{s' \in \hat{S}} (r - s')$$

If $\hat{S}$ is a permutation of $S||[n, ..., m]$, then H is the zero-polynomial. To prove this, the client generates a Prio proof that $H(r) = 0$ for an r that was randomly chosen by the servers. r is generated using the extension to Prio from Davis et al [30].

Efficiency. The sorting proof is the most compute efficient, and, when $[n, m]$ is small and $|S|$ is large, it sends less data than the bit-splitting proof. However, it uses significantly more memory than the bit-splitting proof. This is because the Prio proofs for the permutation argument perform a Fast-Fourier Transform over $|S| + m - n$ values which are materialized in RAM.

A.3 Commitment Proofs using Bulletproofs

The *commitment proof* is a novel modification to Bulletproofs which adapts it for use over secret shares [22]. These proofs are extremely efficient in communication, at the cost of higher compute and memory requirements. This may be desirable in deployments with more powerful clients (e.g. Raspberry Pis) or low-bandwidth connections.

Recall Bulletproofs allows a prover who knows a secret value v to convince a verifier that $v \in [0, 2^b)$ for some b and that a public Pedersen commitment opens to v, without revealing any other information about v (Section 2.4). Like the bit-splitting proof, these proofs can show that a value v is in an arbitrary interval $[n, m]$ by proving that $v - n \in [0, 2^b)$ and $m - v \in [0, 2^b)$ where $b = \lceil \log_2(m - n) \rceil$ (Section A.1).

Bulletproofs cannot be directly used by the system because it does not conduct proofs over secret shares. The commitment proof modifies Bulletproofs to operate over secret shared data by leveraging the fact that Pedersen Commitments are also additively homomorphic. This allows the servers to generate commitments to the client's unshared value.

Assume the client wants to prove a value $s \in S$ is in $[n, m]$. To do this, the client proves that both $s - n$ and $m - s$ are in $[0, 2^b)$. Let v be either $s - n$ or $m - s$. The high-level steps for both proofs are:

(1) The servers generate commitments to their share $[v]$ using randomness chosen by the client.
(2) The servers send these commitments to the resource operator, who multiplies them to create a commitment to v.
(3) The client sends the resource operator a Bulletproofs proof, which they can verify using the commitment.

First, we will show how Bulletproofs can be used on secret shares over $|\mathbb{G}|$, and then show how to extend it to arbitrary fields. Assume there are two servers, where the *leader* is the resource operator and the *follower* is the third-party. The client wants to prove that $v \in [0, 2^b)$ (generalizing to many servers is straightforward). First, the client generates secret shares $[v]_1, [v]_2 \in |\mathbb{G}|$ and random values $r_1, r_2 \in |\mathbb{G}|$. The client sends $[v]_1$ and r_1 to the first server, and $[v]_2$ and r_2 to the second server. $\mathbb{G}$ refers to the Pedersen group, and $|\mathbb{G}|$ is the size of the group. Then, each server $i \in \{1, 2\}$ creates a Pedersen commitment to their share as

$$C_1 = g^{[v]_1} h^{r_1}, C_2 = g^{[v]_2} h^{r_2}$$

Note that this commitment is both hiding and binding. The follower sends C_2 to the leader, who multiplies them to produce a commitment to v:

$$\begin{aligned} C = C_1 \cdot C_2 &= g^{[v]_1} h^{r_1} \cdot g^{[v]_2} h^{r_2} \\ &= g^{[v]_1 + [v]_2 \bmod |\mathbb{G}|} h^{r_1 + r_2} \\ &= g^{v} h^{r_1 + r_2} \end{aligned} \tag{1}$$

This follows from the additive homomorphism of Pedersen commitments, and that $[v]_1, [v]_2$ were shares over $|\mathbb{G}|$. Lastly, the client sends a Bulletproof π, that shows C opens to v and that $v \in [0, 2^b)$, which the leader verifies.

The scheme above has two major limitations. First, the client must use large secret shares ($\approx$ 256-bits), because they are generated over $|\mathbb{G}|$. This leads to huge communication cost overhead. For example, using 256-bit shares instead of 64-bit shares adds 2.4 MB of communication overhead when sending 100,000 shares. Ideally, the client could use shares of any bit-width (e.g. 64-bits) to reduce communication. Second, each server must send one commitment per value to the resource operator. With 512-bit commitments, this adds 6.4 MB of communication over 100,000 proofs. We fix both limitations by modifying Bulletproofs.

To allow clients to use secret shares of any bit-width, the commitment proof adds extra bits to Bulletproof's internal bit-splitting argument. The core challenge in supporting secret shares of any bit-width is handling the "carry" bits that can be produced when summing the secret shares. For example, given two secret shares $[v]_1, [v]_2 \in 2^{64}$, their sum either is v or $v + 2^{64}$. Normally, the extra 2^{64} addend is removed by performing computations mod 2^{64}. However, cannot be done in the exponent of a Pedersen commitment.

To fix this, commitment proofs add the extra bits to the Bulletproofs bit-splitting argument. Recall that internally, Bulletproofs

proves that some value $v \in [0, 2^b)$ by showing that there exists a bit decomposition of v in b bits, such that

$$v = v_0 2^0 + v_1 2^1 + ... + v_{b-1} 2^{b-1}$$

We modify this relation to add the extra carry bit c, as:

$$[v]_1 + [v]_2 = v_0 2^0 + v_1 2^1 + ... + v_{b-1} 2^{b-1} + c 2^{64}$$

When there are more than two servers, the idea can be extended by adding additional carry bits.

To fix the second limitation which makes servers send many commitments to the resource operator, the commitment proof compresses them into a single commitment leveraging associativity of the group. To verify an aggregate Bulletproofs proof, the resource operator must exponentiate and multiply commitments to each value. The key insight is that this operation is associative, which means by reordering the multiplications, each server only sends a single value.

A detailed description of the protocol, including proofs of security, are in Appendix B.

Efficiency. The commitment proof is the most communication efficient, with the client only sending $O(\log(b \cdot |S|))$ data for the proofs. However, it is much more compute and memory intensive because it requires group exponentiations for proving and verifying.

B RANGE PROOF SECURITY ANALYSIS

First, we begin by defining zero-knowledge and soundness in our setting.

Definition 1 (Zero-Knowledge). *Assume that the client and at least one server is honest. That is, the adversary controls some proper subset of corrupted servers. We say the protocol is zero-knowledge if for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a simulator Sim, such that for any schedule* $\mathbf{v}$*,*

$$\mathsf{View}_{\mathcal{A}}(\langle C(\mathbf{v}), \mathcal{A}, H\rangle) \approx_C \mathsf{View}_{\mathcal{A}}(\langle Sim, \mathcal{A}\rangle)$$

where C is the client and H are the honest servers.

Definition 2 (Soundness). *Let n be the minimum rate and m be the maximum rate, l be the number of servers, and $\mathbf{v} = [\mathbf{v}]_1 + ... + [\mathbf{v}]_l$, i.e. the sum of the servers shares. Then, if there is some $v \in \mathbf{v}$ such that $v \notin [n, m]$, then the servers will reject with overwhelming probability.*

Weft requires that all servers are honest, and only requires existential soundness (instead of knowledge soundness). If a server is dishonest, then soundness is impossible because servers can always send invalid shares during the aggregation phase. The system only requires existential soundness because the aggregators only care that the value encoded in their shares is valid. These assumptions have been made in prior work non-colluding third-party model [6, 27].

B.1 Bit-Splitting Proof

Theorem B.1. *When the client and at least 1 server is honest, then the bit-splitting proof using Prio is zero-knowledge.*

Proof. The proof is trivial. The simulator samples values $\mathbf{v} \in [n, m]$ and computes the bits of $m - v$ and $v - n$. Then, it calls the simulator for Prio, as defined in Appendix D of their paper [27]. The view of the adversary when interacting with the simulator is exactly the same as the view when interacting with honest parties. Therefore, the bit-splitting proof is zero-knowledge. □

Next, we will prove the bit-splitting proof is sound using a basic lemma:

Lemma B.2. *Let $v, m, n \in \mathbb{F}$, $b = \lceil \log_2(m - n) \rceil$, and $b \leq \log_2(|\mathbb{F}|) - 1$. If $m - v \in [0, 2^b)$ and $v - n \in [0, 2^b)$ where, then $v \in [n, m]$*

Proof. $n \leq v \leq m$ if and only if $0 \leq v - n \leq m - n$. If $0 \leq m - v \leq 2^b$, then either $v - n \leq m - n$ or $v - n \geq (m - n) - 2^b$. Notice that $(m - n) - 2^b$ is large ($> 2^b$) because $(m - n) < 2^b$ and arithmetic in the field is modulo $|\mathbb{F}|$ (i.e. it "wraps around"). Thus, if $0 \leq m - v \leq 2^b$ and $0 \leq v - n \leq 2^b$, then $n \leq v \leq m$. Therefore, $v \in [n, m]$ for arbitrary values n and m using two bit-splitting proofs: one that shows $m - v \in [0, 2^b)$ and one that shows $v - n \in [0, 2^b)$. □

Theorem B.3. *If all servers are honest, then the bit-splitting protocol is sound.*

Proof. Assume towards a contradiction that the servers accept on some $\mathbf{v} = [\mathbf{v}]_1 + ... + [\mathbf{v}]_n$, where some value $v \in \mathbf{v}$ is not in $[n, m]$. Recall that $v \in [n, m]$ if and only if $m - v \in [0, 2^b)$ and $v - n \in [0, 2^b)$. Similarly, some $x \in [0, 2^b)$ if and only if there is a bit-decomposition $x_0, x_1, ..., x_{b-1}$ such that $x_i \in \{0, 1\}$ for all $i \in 0...b - 1$ and $x = 2^0 x_0 + 2^1 x_1 + ... + 2^{b-1} x_{b-1}$.

By the construction of the bit-splitting proof, this means that the servers accepted on some $x \in \{m - v, v - n\}$ where $x \notin [0, 2^b)$. The servers receive secret shares, $[x_0]$, $[x_1]$, ..., $[x_{b-1}]$, and Prio proofs π_i that $p_b(x_i) = x_i(x_i - 1) = 0$ for all $i \in 0...b - 1$. The bit-decomposition x_0, x_1, ..., x_{b-1} must be invalid by the assumption that $x \notin [0, 2^b)$. Either $x \neq 2^0 x_0 + 2^1 x_1 + ... + 2^{b-1} x_{b-1}$ or a bit $x_i \notin \{0, 1\}$ such that the proof π_i accepted, If $x \neq 2^0 x_0 + 2^1 x_1 + ... + 2^{b-1} x_{b-1}$ then $x - 2^0 x_0 + 2^1 x_1 + ... + 2^{b-1} x_{b-1}$ is non-zero, which contradicts the assumption that the servers accept. If $x_i \notin \{0, 1\}$, then $p_b(x_i) = x_i(x_i - 1) \neq 0$ by the fundamental theorem of algebra. However, this contradicts the soundness of Prio, which states that $\mathsf{Prio.Verify}(\pi_i, p_b, [v_i])$ will reject with overwhelming probability. Therefore, such an x cannot exist.

Thus, for some invalid $\mathbf{v}$, the servers will reject with overwhelming probability. □

B.2 Sorting Proof

Theorem B.4. *When the client and at least 1 server is honest, then the sorting proof using Prio is zero-knowledge.*

Proof. The proof is trivial. The simulator samples random values $\mathbf{v} \in [n, m]$, and computes $\hat{\mathbf{v}} = \mathrm{Sort}(\mathbf{v} || [n, n + 1, ..., m - 1, m])$. Then, it calls the simulator for Prio, as defined in Appendix D of their paper [27]. The view of the adversary when interacting with the simulator is exactly the same as the view when interacting with honest parties. Therefore, the bit-splitting proof is zero-knowledge. □

Theorem B.5. *If all servers are honest, then the sorting protocol is sound.*

Proof. Assume towards a contradiction that the servers run the sorting proof on on some $\mathbf{v} = [\mathbf{v}]_1 + ... + [\mathbf{v}]_n$ where some value $v \in \mathbf{v}$ is not in $[n, m]$, and $|\mathbf{F}| >>> |\mathbf{v}+n-m|$ The servers also hold shares $\hat{\mathbf{v}}$.

There are two cases: either $\mathbf{v} \subset \hat{\mathbf{v}}$, or $\mathbf{v} \not\subset \hat{\mathbf{v}}$. If $\mathbf{v} \not\subset \hat{\mathbf{v}}$, then there is some value $v \in \mathbf{v}$ such that $v \notin \hat{\mathbf{v}}$. However, this implies that $\hat{\mathbf{v}}$ is not a permutation of $\mathbf{v}||[n, n+1, ..., m-1, m]$. Therefore, the permutation argument from Bayer et al. will accept with probability $\frac{|\mathbf{v}+n-m|}{\mathbb{F}}$, which is negligible.

If, $\mathbf{v} \subset \hat{\mathbf{v}}$, then $\hat{\mathbf{v}}$ contains a value $v \notin [n, m]$. If $v = \hat{\mathbf{v}}[0]$ or $v = \hat{\mathbf{v}}[\hat{\mathbf{v}} - 1]$, then the servers will always reject. If $v = \hat{\mathbf{v}}[i]$ for some $i \neq 0$ and $i \neq \hat{\mathbf{v}} - 1$, then $\hat{\mathbf{v}}$ is not sorted. Let v at index i be the first value in $\hat{\mathbf{v}}$ which is not in sorted order. If $v - \hat{\mathbf{v}}[i] \notin \{0, 1\}$, the servers will reject with overwhelming probability by the soundness of Prio. Therefore, $v = 0$ and $\hat{\mathbf{v}}[i-1] = |\mathbb{F}|$. This is impossible because $\hat{\mathbf{v}}[i-1]$ must be sorted and $\hat{\mathbf{v}}[0] = n$, so the maximum value of $\hat{\mathbf{v}}[i-1]$ is $n + i - 1 < m < |\mathbb{F}|$ Therefore, the servers will reject with overwhelming probability. □

B.3 Commitment Proof

This section formally describes and proves the zero-knowledge and soundness of the commitment proof protocol. We split the commitment proofs into two parts. First, we describe a zero-knowledge proof *ModBP* which proves that a vector $\mathbf{v} \in [0, 2^b)^m$ for some b given a commitment $V = \mathbf{g}^{\mathbf{v}+\mathbf{c}\cdot 2^n}\mathbf{h}^{\gamma}$ for some $\mathbf{c} \in \{0, 1\}^m$ Then, we describe the n-server commitment proofs protocol, where the servers jointly verify that the values encoded in their shares are in range.

Figure 6 shows the pseudo-code for *ModBP*. We use similar notation to the original Bulletproofs paper. $\mathbb{G}$ is a group of prime order p. For two vectors $\mathbf{v} \in \mathbb{G}^m, \mathbf{w} \in \mathbb{Z}_p^m$, $\mathbf{v}^{\mathbf{w}} = \prod_{i=0}^{m} v[i]^{w[i]}$. Similarly, we write $\langle \mathbf{v}, \mathbf{w} \rangle$ to mean the dot product between $\mathbf{v}$ and $\mathbf{w}$. The modifications compared to the original Bulletproofs protocol are shown in red. Note that the only changes are made to support the extra carry bit from summing the secret shares. Like Bulletproofs, it can be made non-interactive with Fiat-Shamir.

Theorem B.6. *The modified aggregate Bulletproofs range argument has perfect completeness, perfect honest verifier zero-knowledge and computational witness extended emulation.*

Proof. The proof is identical to the proof of Theorem 3 in the bulletproofs paper [22], except it computes all values using $[2^0, ..., 2^b, 2^n]$ in place of $[2^0, ..., 2^n]$. □

Now, using *ModBP* as a building block, we describe the full commitment proofs.

Figure 8 shows the full commitment proofs protocol in the two server case. One server (LeaderServer) is the battery operator, and the other (FollowerServer) is the third-party. With l servers, there is still one Leader, but $l-1$ Followers. Most of the protocol is from *ModBP*, with a few key differences. First, the prover generates commitments $\mathbf{V}$ to $[\mathbf{v}]_1 + [\mathbf{v}]_2$, and blinds γ_1 and γ_2. Second, All challenges are generated using the *GenChallenge* protocol, shown in Figure 7. Third, the leader server computes $\hat{V}$ as required for the verification equation as

$$\hat{V}_1 \cdot \hat{V}_2 = \mathbf{V}_1^{z^2 \cdot \mathbf{z}^m} \mathbf{V}_2^{z^2 \cdot \mathbf{z}^m} = (\mathbf{V}_1 \cdot \mathbf{V}_2)^{z^2 \cdot \mathbf{z}^m} = \mathbf{V}^{z^2 \cdot \mathbf{z}^m}$$

using $\hat{V}_2$ sent from the follower server.

First, we will prove that the challenges generated by GenChallenge are always uniformly distributed.

Lemma B.7. *As long as at least one server is honest, then the protocol GenChallenge will generate uniformly random challenges.*

Proof. The proof is direct from the fact that z is the sum of z_1 and z_2. If z_1 was generated maliciously, z_2 is still uniformly random. Thus, z is uniformly random. Similarly, if z_2 was generated maliciously, z_1 is still uniformly random. Thus, z is uniformly random. □

Now, we will prove the protocol is zero-knowledge.

Theorem B.8. *If the client and at least one server are honest, then the Commitment proofs protocol is zero-knowledge.*

Proof. We will prove that the commitment proof is zero-knowledge. We want to show that an adversary which corrupts either the leader or follower servers cannot learn anything about the clients values $\mathbf{v}$. To do this, we will show that the transcript between the adversary and the honest parties can be simulated without knowledge of $\mathbf{v}$.

The proof outline is to define a real vs ideal world experiment. In the real world, the Adversary interacts with the real protocol. In the ideal world, the Adversary interacts with the simulator. The proof shows that the view of the adversary in both worlds is indistinguishable, proving that the view of the adversary leaks no information about $\mathbf{v}$.

We present the proof when there are two servers (so either the Leader or Follower is corrupted, but not both). Then we will show how to extend the proof to an arbitrary number of servers, where any proper subset are corrupted.

In the first case, the Leader is corrupted. The simulator Sim_1 which interacts with the corrupted Leader is defined in Figure 9. We will now argue that the view of the adversary $\mathcal{A}$ when interacting with Sim_1 is indistinguishable the view of $\mathcal{A}$ interacting with the honest client and follower. We prove this using a Hybrid argument.

Hybrid 1: In the first hybrid, we replace the random $\tilde{\mathbf{v}}$ in Sim_1 with actual data $\mathbf{v}$. Define H_1 as exactly the same as Sim_1, except that it uses the actual data $\mathbf{v}$. Assume that there exists an adversary $\mathcal{B}$ which distinguishes between transcripts of adversary $\mathcal{A}$ interacting with Sim_1 or H_1 with non-negligible probability. We will use this to build a distinguisher between the Pedersen commitments $V = g^{\tilde{\mathbf{v}}}h^{\gamma}$ and $V' = g^{\mathbf{v}}h^{\gamma'}$ with non-negligible probabilities. The code for the distinguisher is very similar to Sim_1. It differs from Sim_1 in two ways. First, it does not compute $[v]_2$ or γ_2. Second, it computes $\hat{V}_2 = V^{z^2 \cdot \mathbf{z}^m} / (\mathbf{g}^{[\mathbf{v}]_1}\mathbf{h}^{\gamma_1})^{z^2 \cdot \mathbf{z}^m}$. On input V, the distinguisher produces a view identical to Sim_1, and on input V' the distinguisher produces a view identical to H_1. This is because $\hat{V}_2 = V^{z^2 \cdot \mathbf{z}^m} / (\mathbf{g}^{[\mathbf{v}]_1}\mathbf{h}^{\gamma_1})^{z^2 \cdot \mathbf{z}^m} = \mathbf{g}^{\mathbf{v}-[\mathbf{v}]_1}\mathbf{g}^{\gamma_2} = \mathbf{g}^{[\mathbf{v}]_2}\mathbf{h}^{\gamma_2}$. The distinguisher outputs the result of $\mathcal{B}$ on the constructed transcript. The probability that the distinguisher succeeds equals the probability that $\mathcal{B}$ succeeds, and therefore we have a distinguisher between two Pedersen commitments $V = g^{\tilde{\mathbf{v}}}h^{r}$ and $V' = g^{\mathbf{v}}h^{r'}$, contradicting the hiding property of the commitments. Therefore, $\mathcal{B}$ does not exist, and the transcripts of Sim_1 and H_1 must be indistinguishable.

$\mathcal{P}(\mathbf{v} \in (0..2^b)^m, \gamma \in \mathbb{Z}_p^m, c \in \{0,1\}^m, \mathbf{g} \in \mathbb{G}^{(b+1)\cdot m}, \mathbf{h} \in \mathbb{G}^{(b+1)\cdot m}, g \in \mathbb{G}, h \in \mathbb{G}, \mathbf{V} = g^{\mathbf{v}+\mathbf{c}\cdot 2^n} h^{\gamma})$
$\mathbf{a}_L = \text{BitDecomp}(\mathbf{v}, \mathbf{c}, b)$ // $v[j] = \langle [2^0, 2^1, ..., 2^b, 2^n], \mathbf{a}_L[j-1\cdot(b+1) : j\cdot(b+1)] \rangle$ forall $j \in 1..m$
$\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}$
$\alpha \xleftarrow{\$} \mathbb{Z}_p$
$A = h^{\alpha} \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$
$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^{(b+1)\cdot m}$
$\rho \xleftarrow{\$} \mathbb{Z}_p$
$S = h^{\rho} \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$
$\mathcal{P} \rightarrow \mathcal{V}$: A, S
$\mathcal{V} \rightarrow \mathcal{P}$: y, z
$\mathbf{y} = [y^0, y^1, ..., y^{(b+1)\cdot m}] \in \mathbb{Z}_p^{(b+1)\cdot m}$
$l(X) = (\mathbf{a}_L - z \cdot \mathbf{1}^{(b+1)\cdot m}) + \mathbf{s}_L \cdot X$
$r(X) = \mathbf{y} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{(b+1)\cdot m} + \mathbf{s}_R \cdot X) + \sum_{j=1}^{m} z^{1+j} \cdot (\mathbf{0}^{(j-1)\cdot(b+1)} || [2^0, 2^1, ..., 2^b, 2^n] || \mathbf{0}^{(m-j)\cdot(b+1)})$
$t(X) = \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2$
$\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_p$
$T_1 = g^{t_1} h^{\tau_1}, T_2 = g^{t_2} h^{\tau_1}$
$\mathcal{P} \rightarrow \mathcal{V}$: T_1, T_2
$\mathcal{V} \rightarrow \mathcal{P}$: x
$\mathbf{l} = l(x) = \mathbf{a}_L - z \cdot \mathbf{1}^{b+1} + \mathbf{s}_L \cdot x$
$\mathbf{r} = r(x) = \mathbf{y} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{b+1} + \mathbf{s}_R \cdot x) + z^2 \cdot [2^0, 2^1, ..., 2^b, 2^n]$
$\hat{t} \leftarrow \langle \mathbf{l}, \mathbf{r} \rangle$
$\tau_x \leftarrow \tau_2 \cdot x^2 + \tau_1 \cdot x + \sum_{j=1}^{m} z^{1+j} \cdot \gamma[j]$
$\mu = \alpha + \rho \cdot x$
$\mathcal{P} \rightarrow \mathcal{V}$: $\tau_x, \mu, \hat{t}$
Run inner-product argument to prove $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$

$\mathcal{V}(\mathbf{g} \in \mathbb{G}^{(b+1)\cdot m}, \mathbf{h} \in \mathbb{G}^{(b+1)\cdot m}, g \in \mathbb{G}, h \in \mathbb{G}, \mathbf{V} \in \mathbb{G}^m)$
$\mathcal{P} \rightarrow \mathcal{V}$: A, S
$y, z \xleftarrow{\$} \mathbb{Z}_p$
$\mathcal{V} \rightarrow \mathcal{P}$: y, z
$\mathcal{P} \rightarrow \mathcal{V}$: T_1, T_2
$x \xleftarrow{\$} \mathbb{Z}_p$
$\mathcal{V} \rightarrow \mathcal{P}$: x
$\mathcal{P} \rightarrow \mathcal{V}$: $\tau_x, \mu, \hat{t}$
$h_i' \leftarrow h_i^{(y^{-i+1}])} \forall i \in [1, b+1]$
$\delta \leftarrow (z - z^2) \cdot \langle \mathbf{1}^{(b+1)\cdot m}, \mathbf{y}^{(b+1)\cdot m} \rangle - \sum_{j=1}^{m} z^{j+2} \cdot \langle \mathbf{1}^{b+1}, [2^0, ..., 2^b, 2^n] \rangle$
$\hat{V} \leftarrow \mathbf{V}^{z^2 \cdot \mathbf{z}^m}$
$g^{\hat{t}} h^{\tau_x} \stackrel{?}{=} g^{\delta} \cdot \hat{V} \cdot T_1^x \cdot T_2^{x^2}$
$P \leftarrow A \cdot S^x \cdot \mathbf{g}^{-z} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}} \prod_{j=1}^{m} (\mathbf{h}'[(j-1)\cdot(b+1) : j\cdot(b)])^{z^{j+1} \cdot [2^0, 2^1, ..., 2^b, 2^n]}$
$P \stackrel{?}{=} h^{\mu} \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{h}')^{\mathbf{r}}$
Verify inner-product argument for $\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$
If all checks succeed, output Accept. Otherwise, output Reject.

Figure 6: Pseudo-code for the the Modified Bulletproofs Range Argument

Hybrid 2: Now, we will show that H_1 is indistinguishable from the real protocol. Assume that there is an adversary $\mathcal{B}$ which distinguishes between accepting transcripts of an adversary $\mathcal{A}$ interacting with H_1 or the real protocol. We will define an adversary which can distinguish between transcripts generated by $ModBP.Sim(V)$ and $ModBP.Prove(v, r, V)$. The code for the distinguisher is similar to H_1, except it uses the input transcript instead of running

Prover
$Leader \rightarrow \mathcal{P} : z_1$
$Follower \rightarrow \mathcal{P} : z_2$
$z = z_1 + z_2$
$\mathcal{P} \rightarrow Leader : z$

FollowerServer
$z_2 \xleftarrow{\$} \mathbb{Z}_p$
$Follower \rightarrow \mathcal{P}: z_2$
$\mathcal{P} \rightarrow Follower: z$
$Follower \rightarrow Leader: z_2$

LeaderServer
$z_1 \xleftarrow{\$} \mathbb{Z}_p$
$Leader \rightarrow \mathcal{P} : z_1$
$\mathcal{P} \rightarrow Leader : z$
$Follower \rightarrow Leader : z_2$
$z \stackrel{?}{=} z_1 + z_2$

Figure 7: Protocol GenChallenge, which allows the servers to jointly generate random challenges

$ModBP.Sim(V)$. Just as in Hybrid 1, $\hat{V}_2$ is the same value as required for the real protocol, even though it is computed differently. Therefore, the distinguisher produces a view of $\mathcal{A}$ that is distributed identically for either H_1 or the real protocol. Output the result of $\mathcal{B}$ on the constructed transcript. The probability that the distinguisher succeeds equals the probability that $\mathcal{B}$ succeeds, and therefore we have a distinguisher between transcripts generated by $ModBP.Sim(V)$ and $ModBP.Prove(v, r, V)$, which violates honest-verifier zero-knowledge of $ModBP$. Therefore, $\mathcal{B}$ does not exist, and the transcripts between the adversary interacting with H_1 and the real protocol are indistinguishable.

Thus, we have proven that a transcript of any polynomial-time adversary when interacting with the simulator Sim_1 is indistinguishable from the view of the adversary when interacting with the real protocol. Therefore, the protocol is zero-knowledge when the Leader is corrupted.

Next, we will prove that the protocol is zero-knowledge when the Follower is corrupted. We define the simulator Sim_2 in Figure 9. We will now argue that the view of the adversary $\mathcal{A}$ when interacting with Sim_2 is indistinguishable the view of $\mathcal{A}$ interacting with the honest client and follower. Again, we will use a hybrid argument.

Hybrid 1: As in the Leader case, in the first hybrid, we replace the random $\tilde{\mathbf{v}}$ with actual data $\mathbf{v}$. Define H_1 as exactly the same as Sim_2, except that it uses the actual data $\mathbf{v}$. Assume that there exists an adversary $\mathcal{B}$ which distinguishes between transcripts of adversary $\mathcal{A}$ interacting with Sim_2 or H_1 with non-negligible probability. We will use this to build a distinguisher between the Pedersen commitments $V = g^{\tilde{\mathbf{v}}}h^r$ and $V' = g^{\mathbf{v}}h^{r'}$ with non-negligible probabilities. The code for the distinguisher is very similar exactly the same as Sim_2, except it runs $BP.Sim$ on the input commitment. On input V, the distinguisher produces a view identical to Sim_1, and on input V' the distinguisher produces a view identical to H_1. Output the result of $\mathcal{B}$ on the constructed view. The probability that the distinguisher succeeds equals the probability that $\mathcal{B}$ succeeds, and therefore we have a distinguisher between two Pedersen commitments $V = g^{\tilde{\mathbf{v}}}h^r$ and $V' = g^{\mathbf{v}}h^{r'}$, contradicting the hiding property of the commitments. Therefore, $\mathcal{B}$ does not exist, and the transcripts of Sim_1 and H_1 must be indistinguishable.

Hybrid 2: Now, we will show that H_1 is indistinguishable from the real protocol. Assume that there is an adversary $\mathcal{B}$ which distinguishes between accepting transcripts of an adversary $\mathcal{A}$ interacting with H_1 or the real protocol. We will define an adversary which can distinguish between transcripts generated by $ModBP.Sim(V)$ and $ModBP.Prove(v, r, V)$. The code for the distinguisher is to H_1, except it uses the transcript given as input instead of running $ModBP.Sim$. The distinguisher produces a view of $\mathcal{A}$ that is distributed identically for either H_1 or the real protocol. Output the result of $\mathcal{B}$ on the constructed transcript. The probability that the distinguisher succeeds equals the probability that $\mathcal{B}$ succeeds, and therefore we have a distinguisher between transcripts generated by $ModBP.Sim(V)$ and $ModBP.Prove(v, r, V)$, which violates honest-verifier zero-knowledge of $ModBP$. Therefore, $\mathcal{B}$ does not exist, and the transcripts between the adversary interacting with H_1 and the real protocol are indistinguishable. Thus, the protocol is zero-knowledge when the follower is corrupted.

Because the protocol is zero-knowledge when both the Follower or Leader are corrupted, the protocol is zero-knowledge.

Lastly, we show how to extend the proof to the n party setting, where any proper subset are corrupted. There is always a single leader, so there are two cases, either the subset includes the leader or it doesn't. In the case the corrupted subset doesn't include the Leader, then the simulator is exactly as Sim_2, except that it also computes additional y_i, z_i, and x_i for each follower such that they are consistent with y, z, and x from $BP.Sim$. The hybrids from the

$\mathcal{P}(\mathbf{v} \in (0..2^b)^m)$
$[\mathbf{v}]_1 \xleftarrow{\$} (0..2^n)^m$, $[\mathbf{v}]_2 = \mathbf{v} - [\mathbf{v}]_1$
$\gamma_1 \xleftarrow{\$} \mathbb{Z}_p^m$, $\gamma_2 \xleftarrow{\$} \mathbb{Z}_p^m$
$\mathbf{a}_L = \text{BitDecomp}([\mathbf{v}]_1 + [\mathbf{v}]_2, b)$
$\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}$
$\alpha \xleftarrow{\$} \mathbb{Z}_p$
$A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}$
$\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^{(b+1)\cdot m}$
$\rho \xleftarrow{\$} \mathbb{Z}_p$
$S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$
$\mathcal{P} \rightarrow$ Leader: $[\mathbf{v}]_1, \gamma_1, A, S$
$\mathcal{P} \rightarrow$ Follower: $[\mathbf{v}]_2, \gamma_2$
Run *GenChallenge* to generate y, z
$\mathbf{y} = [y^0, y^1, ..., y^{(b+1)\cdot m}] \in \mathbb{Z}_p^{(b+1)\cdot m}$
$l(X) = (\mathbf{a}_L - z \cdot \mathbf{1}^{(b+1)\cdot m}) + \mathbf{s}_L \cdot X$
$r(X) = \mathbf{y} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{(b+1)\cdot m} + \mathbf{s}_R \cdot X) + \sum_{j=1}^m z^{1+j}$
$\cdot(\mathbf{0}^{(j-1)\cdot(b+1)} || [2^0, 2^1, ..., 2^b, 2^n] || \mathbf{0}^{(m-j)\cdot(b+1)})$
$t(X) = \langle l(X), r(X) \rangle = t_0 + t_1 \cdot X + t_2 \cdot X^2$
$\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_p$
$T_1 = g^{t_1} h^{\tau_1}$, $T_2 = g^{t_2} h^{\tau_1}$
$\mathcal{P} \rightarrow$ Leader: T_1, T_2
Run *GenChallenge* to generate y, z
$\mathbf{l} = l(x) = \mathbf{a}_L - z \cdot \mathbf{1}^{b+1} + \mathbf{s}_L \cdot x$
$\mathbf{r} = r(x) = \mathbf{y} \circ (\mathbf{a}_R + z \cdot \mathbf{1}^{b+1} + \mathbf{s}_R \cdot x) + z^2 \cdot [2^0, 2^1, ..., 2^b, 2^n]$
$\hat{t} \leftarrow \langle \mathbf{l}, \mathbf{r} \rangle$
$\tau_x \leftarrow \tau_2 \cdot x^2 + \tau_1 \cdot x + \sum_{j=1}^m z^{1+j} \cdot \gamma[j]$
$\mu = \alpha + \rho \cdot x$
$\mathcal{P} \rightarrow \mathcal{V}$: $\tau_x, \mu, \hat{t}$
Run inner-product argument to prove $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$

Follower()
$\mathcal{P} \rightarrow$Follower: $[v]_2, \gamma_2$
Run *GenChallenge* to generate all challenges
$\mathbf{V}_2 \leftarrow g^{[\mathbf{v}]_2} h^{\gamma_2}$
$\hat{V}_2 \leftarrow \mathbf{V}_2^{z^2 \cdot \mathbf{z}^m}$
Send $\hat{V}_2$ to LeaderServer
Accept if Leader Accepts. Reject otherwise.

Leader()
$\mathcal{P} \rightarrow$ Leader: $[v]_1, \gamma_1, A, S$
Run *GenChallenge* to generate y, z
$\mathcal{P} \rightarrow$ Leader: T_1, T_2
Run *GenChallenge* to generate x
$\mathcal{P} \rightarrow$ Leader: $\tau_x, \mu, \hat{t}$
Follower $\rightarrow$ Leader: $\hat{V}_2$
$\mathbf{V}_1 \leftarrow g^{[\mathbf{v}]_1} h^{\gamma_1}$
$\hat{V}_1 \leftarrow \mathbf{V}_1^{z^2 \cdot \mathbf{z}^m}$
$h'_i \leftarrow h_i^{(y^{-i+1})} \forall i \in [1, b+1]$
$\delta \leftarrow (z - z^2) \cdot \langle \mathbf{1}^{(b+1)\cdot m}, \mathbf{y}^{(b+1)\cdot m} \rangle - \sum_{j=1}^m z^{j+2}$
$\cdot\langle \mathbf{1}^{b+1}, [2^0, ..., 2^b, 2^n] \rangle$
$\hat{V} \leftarrow \hat{V}_1 \cdot \hat{V}_2$
$g^{\hat{t}} h^{\tau_x} \stackrel{?}{=} g^\delta \cdot \hat{V} \cdot T_1^x \cdot T_2^{x^2}$
$P \leftarrow A \cdot S^x \cdot \mathbf{g}^{-z} \cdot (\mathbf{h}')^{z \cdot \mathbf{y}}$
$\cdot \prod_{j=1}^m (\mathbf{h}'[(j-1)\cdot(b+1) : j\cdot(b)])^{z^{j+1} \cdot [2^0, 2^1, ..., 2^b, 2^n]}$
$P \stackrel{?}{=} h^\mu \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{h}')^{\mathbf{r}}$
Verify inner-product argument for $\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$
Accept if all check succeed. Reject otherwise.

Figure 8: Commitment Proofs Protocol. It is parameterized by $g, h \in \mathbb{G}, \mathbf{g}, \mathbf{h} \in \mathbb{G}^m$

2-server corrupted follower proof then follow naturally. In the case the corrupted subset includes the Leader, then the simulator is similar to Sim_1. However, the simulator also computes additional y_i, z_i, and x_i for each follower such that they are consistent with y, z, and x from *BP.Sim*, and it receives $\hat{V}_i$ values from each corrupted follower. Then, the proof is straightforward from the proof of the two-server case. □

Now, we argue that the Commitment proofs protocol is sound.

Theorem B.9. *When all servers are honest, the commitment proofs protocol is sound.*

Proof. Assume that the commitment proofs protocol is not sound. Then there is an adversary $\mathcal{A}$ that, when interacting with the servers, can convince the servers to accept some shares such that $v = \sum [\mathbf{v}]_i$ and $v \mod 2^n \notin [0..2^b)^m$. We will use $\mathcal{A}$ to break the soundness of *ModBP*. Define an adversary $\mathcal{B}$ for *ModBP* which interacts with $\mathcal{A}$ and the *ModBP* verifier $\mathcal{V}$. When interacting with $\mathcal{A}$, $\mathcal{B}$ acts as all the servers. When $\mathcal{B}$ receives $[\mathbf{v}]_i$ and γ_i for $i = 0, ..., l$, it computes $V = g^{\sum [\mathbf{v}]_i} h^{\sum \gamma_i}$, and forwards it to $\mathbf{V}$. $\mathcal{B}$ forwards $A, S, T_1, T_2, \tau_x, \mu, \hat{t}$, and the IPA messages from $\mathcal{A}$ to $\mathcal{V}$. When it gets a challenge x from $\mathcal{V}$, it computes corresponding pieces x_i for the leader and each follower, and sends them to $\mathcal{A}$.

We will show that $\mathcal{V}$ accepts when interacting with $\mathcal{B}$ exactly when honest servers accept when interacting with $\mathcal{A}$. Notice that the verification conditions in the Leader are exactly that of the *ModBP* verifier $\mathcal{V}$, except it computes $\hat{V}$ using information from the followers. The value computed from $\hat{V}$ is exactly $V^{z^2 \cdot \mathbf{z}^m}$ that is computed by $\mathcal{V}$. All other values given to $\mathcal{V}$ are the same as those given to the Leader. Therefore, $\mathcal{V}$ must accept when the servers accept. Thus, $\mathcal{B}$ can convince $\mathcal{V}$ of false *ModBP* proofs, which violates the soundness of *ModBP*. Therefore, $\mathcal{A}$ does not exist, and the protocol is sound. □

B.3.1 Efficiency. The protocol presented in Figure 8 has the client send blinds $\gamma_1, \gamma_2 \in \mathbb{Z}_p^m$ to each server. In practice, this leads to huge communication blow-up, because p is large. Therefore, in

$Sim_1()$
$\tilde{\mathbf{v}} \xleftarrow{\$} (0..2^b)^m$
$\tilde{\gamma_1} \xleftarrow{\$} \mathbb{Z}_p^m$
$\tilde{\gamma_2} \xleftarrow{\$} \mathbb{Z}_p^m$
$[\mathbf{v}]_1 \xleftarrow{\$} (0..2^n)^m$, $[\mathbf{v}]_2 \xleftarrow{\$} \mathbf{v} - [\mathbf{v}]_1$
$V = \mathbf{g}^{[\mathbf{v}]_1}\mathbf{h}^{\gamma_1}\mathbf{g}^{[\mathbf{v}]_2}\mathbf{h}^{\gamma_2}$
$(A, S, y, z, x, \tau_x, \mu, \hat{t}, \text{ipa\_tr}) \leftarrow ModBP.Sim(V)$
Send $[v]_1, \gamma_1$ to LeaderServer
Send A, S to LeaderServer
Receive y_1, z_1 from LeaderServer
$y_1, z_1 \leftarrow \mathbb{Z}_p$
$z_2 = z - z_1$, $y_2 = y - y_1$
Send z, y, z_2, y_2 to LeaderServer
Receive x_1 from LeaderServer
$x_1 \leftarrow \mathbb{Z}_p$
$x_2 = x - x_1$
Send x, x_2 to LeaderServer
$\hat{V}_2 \leftarrow (\mathbf{g}^{[\mathbf{v}]_2}\mathbf{h}^{\gamma_2})^{z^2 \cdot \mathbf{z}^m}$
Send $\hat{V}_2$ to LeaderServer
Compute and send parts of the challenges for the IPA transcript ipa_tr.
Output Accept if the Adversary Accepts, output reject otherwise.

$Sim_2()$
$\tilde{\mathbf{v}} \xleftarrow{\$} (0..2^b)^m$
$\tilde{\gamma_1} \xleftarrow{\$} \mathbb{Z}_p^m$
$\tilde{\gamma_2} \xleftarrow{\$} \mathbb{Z}_p^m$
$[\mathbf{v}]_1 \xleftarrow{\$} (0..2^n)^m$, $[\mathbf{v}]_2 \xleftarrow{\$} \mathbf{v} - [\mathbf{v}]_1$
$V = \mathbf{g}^{[\mathbf{v}]_1}\mathbf{h}^{\gamma_1}\mathbf{g}^{[\mathbf{v}]_2}\mathbf{h}^{\gamma_2}$
$(A, S, y, z, x, \tau_x, \mu, \hat{t}, \text{ipa\_tr}) \leftarrow ModBP.Sim(V)$
Send $[\mathbf{v}]_2, \gamma_2$ to FollowerServer
Receive y_2, z_2, x_2 from FollowerServer
$z_1 = z - z_2$, $y_1 = y - y_2$, $x_1 = x - x_2$
Send y, z, x, y_1, z_1, x_1 to FollowerServer
Receive $\hat{V}_2$ from FollowerServer
Compute and send parts of the challenges for the IPA transcript ipa_tr.
Output Accept if the LeaderServer verification conditions hold, output Reject otherwise.

Figure 9: Simulators for the proof that Commitment Proofs are zero-knowledge in the two server setting.

practice the leader and followers generate γ_i using a pseudo-random generator with a seed sent by the client. As long as the pseudo-random generator is secure, the protocol remains zero-knowledge and sound. Lastly, the protocol requires all servers participate in challenge generation. Using a verifiable random function or random oracle, the amount of interaction between the servers and client can be reduced.