Armadillo: Robust Secure Aggregation for Federated Learning with Input Validation

Yiping Ma Yue Guo Harish Karthikeyan Antigoni Polychroniadou

The secure aggregation problem: motivation

People's habit of setting password?

Browsing statistics?

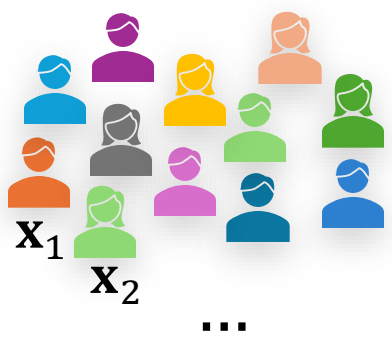
Average blood pressure in young population?

Transaction patterns of the general public?

The secure aggregation problem

Client i has an input $\mathbf{x}_i$

Goal: Learn only $f(\mathbf{x}_1, \mathbf{x}_2, \dots)$



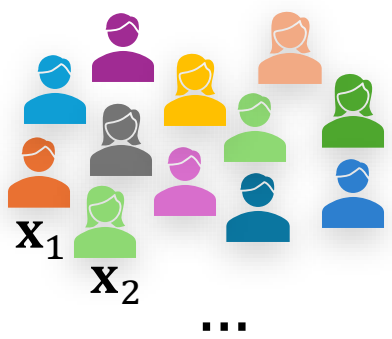
Correctness: if everyone follows the protocol, then the server gets $f(\mathbf{x}_1, \mathbf{x}_2, \dots)$

Privacy: the server only learns $f(\mathbf{x}_1, \mathbf{x}_2, \dots)$ but nothing else

The secure aggregation problem

Client i has an input $\mathbf{x}_i$

Goal: Learn only $f(\mathbf{x}_1, \mathbf{x}_2, \dots)$



Robustness: even when some clients misbehave, the server can always learn $f(\mathbf{x}_1, \mathbf{x}_2, \dots)$ for those valid $\mathbf{x}_i$'s.

Challenges and goals

Clients have limited computation and bandwidth

Efficiency

Client failures happen quite often and unpredictably

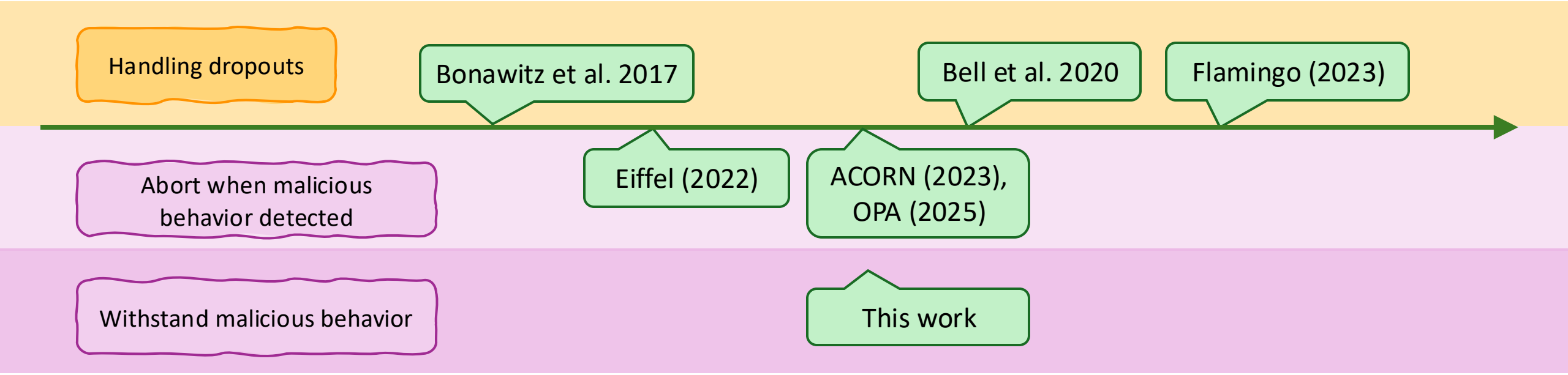
Tolerating dropouts

Malicious clients can actively react in the execution

Withstand malicious behavior

What progress has been made so far?

→ means more efficient



This work

- Robustness guarantee: server only learns the sum of client inputs with valid L_2, L_∞ norms
- 3 rounds of server-client communication
- Communication model:
  - Clients can only directly talk to the server
  - Assuming PKI, a client can talk with other clients via the server with their messages encrypted
- Threat model:
  - Server is semi-honest (trying to violate privacy)
  - Clients can be arbitrarily malicious (trying to disrupt the aggregation)
  - A (static) bounded fraction of malicious clients

Key idea: **co-design** aggregation and proof

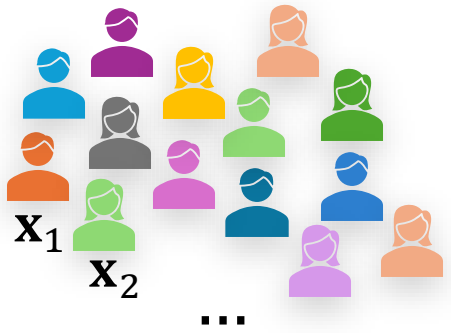
- A secure aggregation protocol with very simple arithmetic computation
- Each client can write their proof statements as inner-product relation

An inner-outer aggregation paradigm

Tool: key-and-message homomorphic encryption

Regev's encryption: linear operations

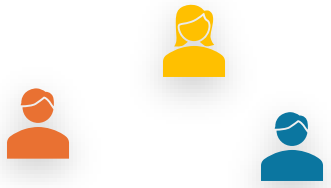
"Outer"



$\mathrm{Enc}(\mathbf{s}_i, \mathbf{x}_i) = \mathbf{y}_i$

$\sum_i \mathbf{y}_i = \mathrm{Enc}(\sum_i \mathbf{s}_i, \sum_i \mathbf{x}_i)$

"Inner"

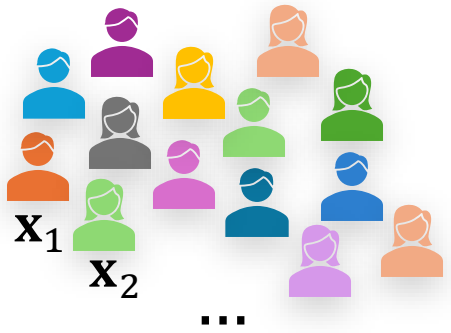


A share of $\mathbf{s}_i$

Secure aggregation on $\mathbf{s}_i$

Easy to handle dropouts

"Outer"

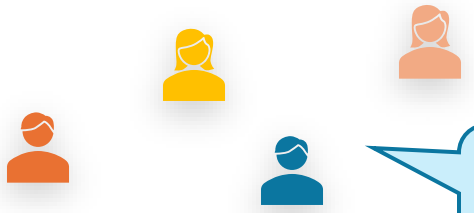


If a client drops,
no impact on subsequent steps

$$\mathrm{Enc}(\mathbf{s}_i, \mathbf{x}_i) = \mathbf{y}_i$$

$$\sum_i \mathbf{y}_i = \mathrm{Enc}\left(\sum_i \mathbf{s}_i, \sum_i \mathbf{x}_i\right)$$

"Inner"



If a committee-client drops,
majority can continue

Resisting disruption

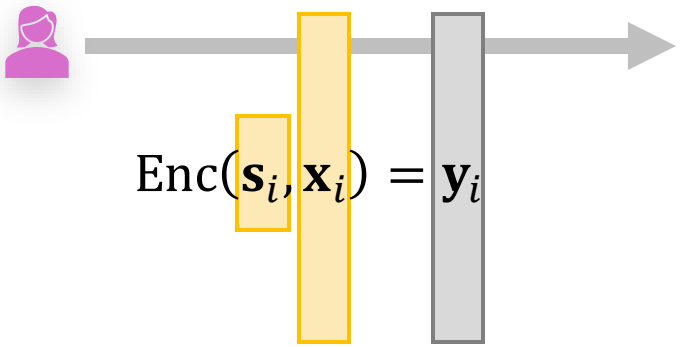
"Encryption is correctly computed"

Inner-product relation

"Shares of $\mathbf{s}_i$ are consistent with its commitment"

*The error vector in Regev's encryption needs to be committed and proven norm as well

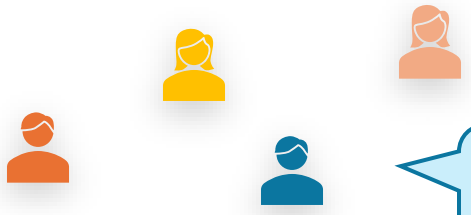
Commit($\mathbf{s}_i$)
Commit($\mathbf{x}_i$)



$$\mathrm{Enc}(\mathbf{s}_i, \mathbf{x}_i) = \mathbf{y}_i$$

Batch check all the proofs

"Inner"



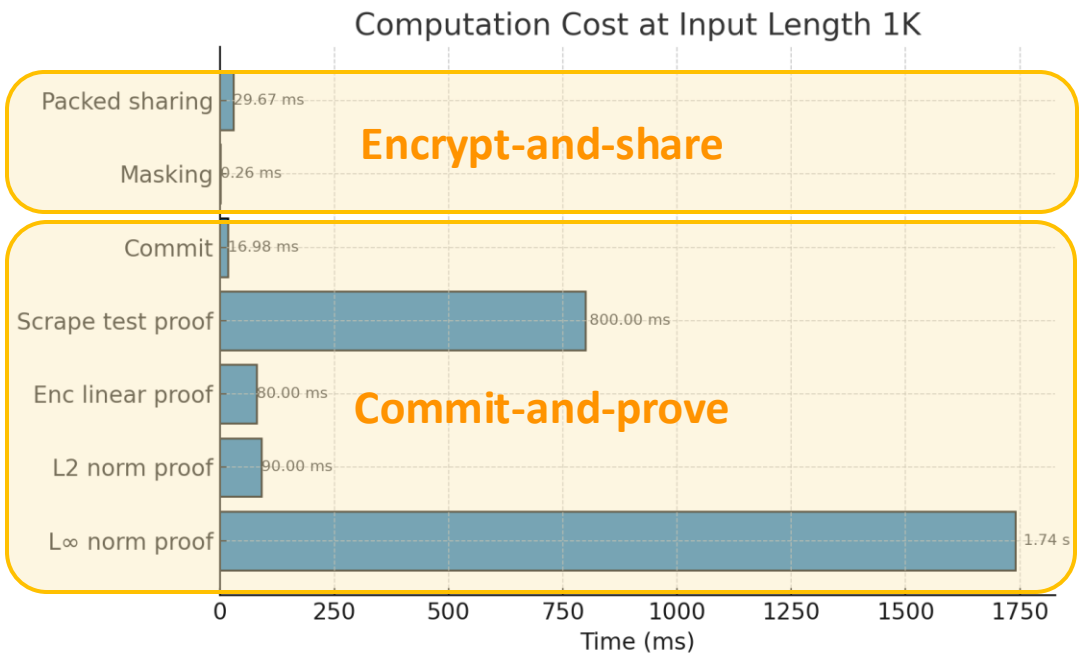
Help with aggregating $\mathbf{s}_i$ for those who send valid proofs

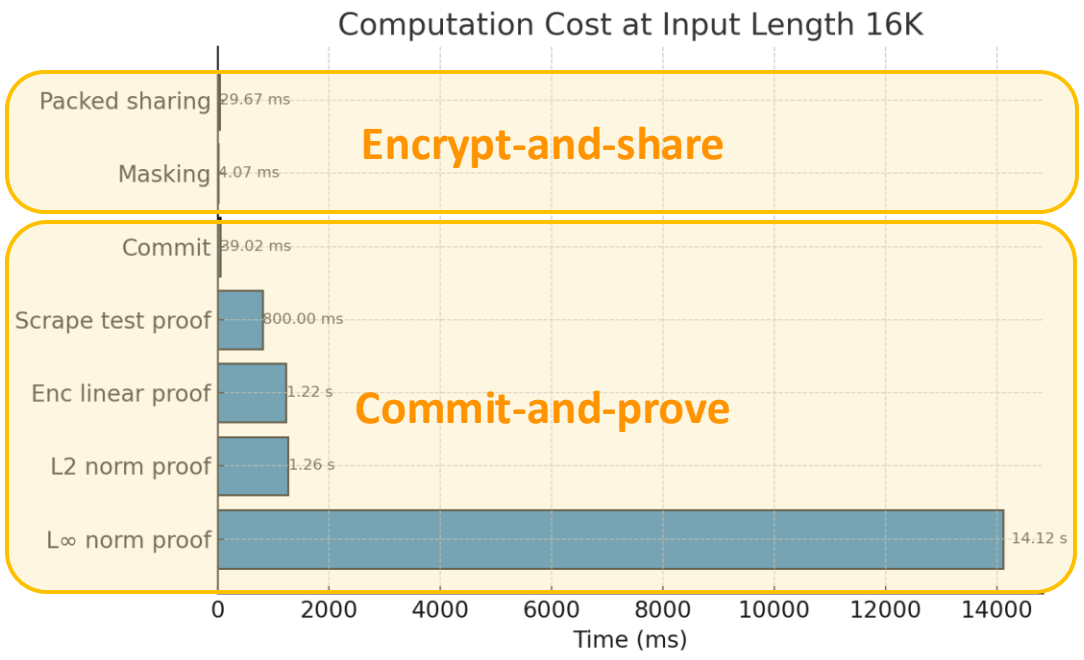
Evaluation setup

- Client:
  - MacBook Air, Apple M2, 8GB RAM
  - Benchmarks using a single thread
- Server:
  - Google Cloud Platform
  - Instance Type: n2-standard-8 (8 vCPUs, 32 GB RAM)
  - Benchmarks using 8 threads

Robustness can be achieved with reasonable cost

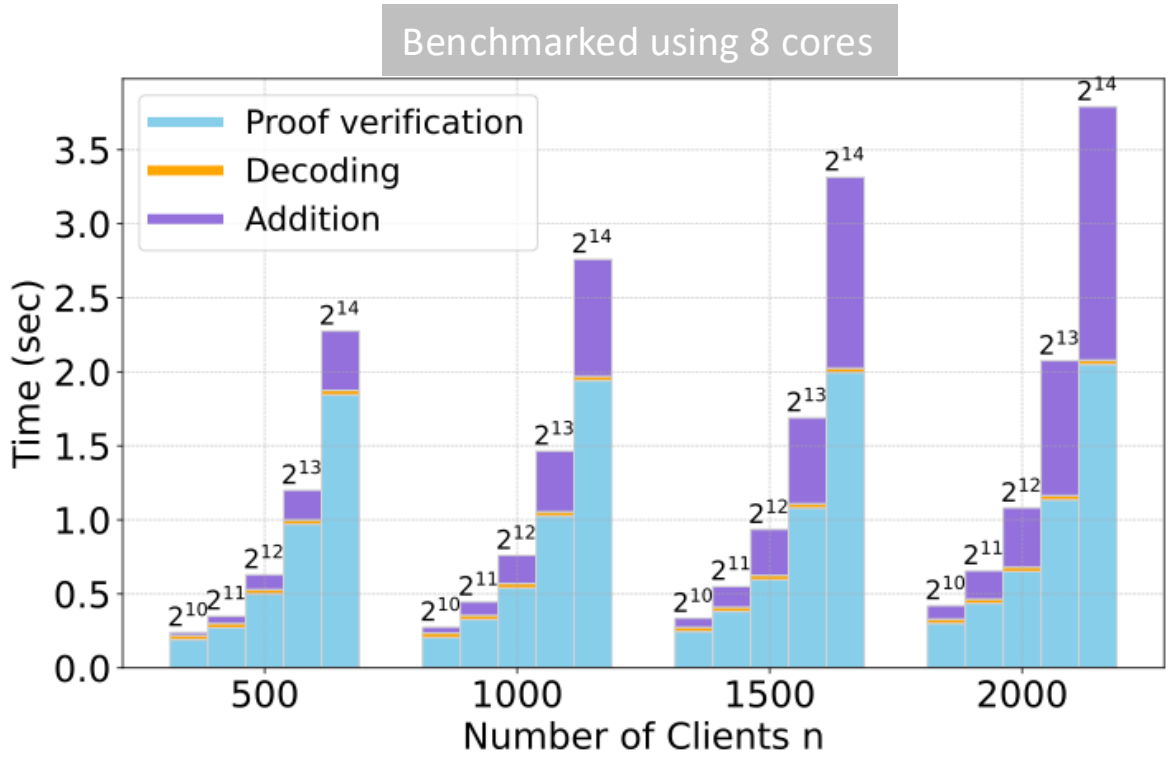
- Client: without proof, only on the order of millisecond; with proof, 1-10 seconds depending on input size





Robustness can be achieved with reasonable cost

- Server computation is highly parallelizable
- Batch verification: computation scales only logarithmic with number of clients



Summary

- Making secure aggregation robust is practically feasible
- What's next?
  - Reducing cost for proof of infinity norm (the current bottleneck)
  - Other useful metrics to verify
  - Privacy against a dynamic adversary

Thanks!