PwoP: Intrusion-Tolerant and Privacy-Preserving Sensor Fusion

Chenglu Jin

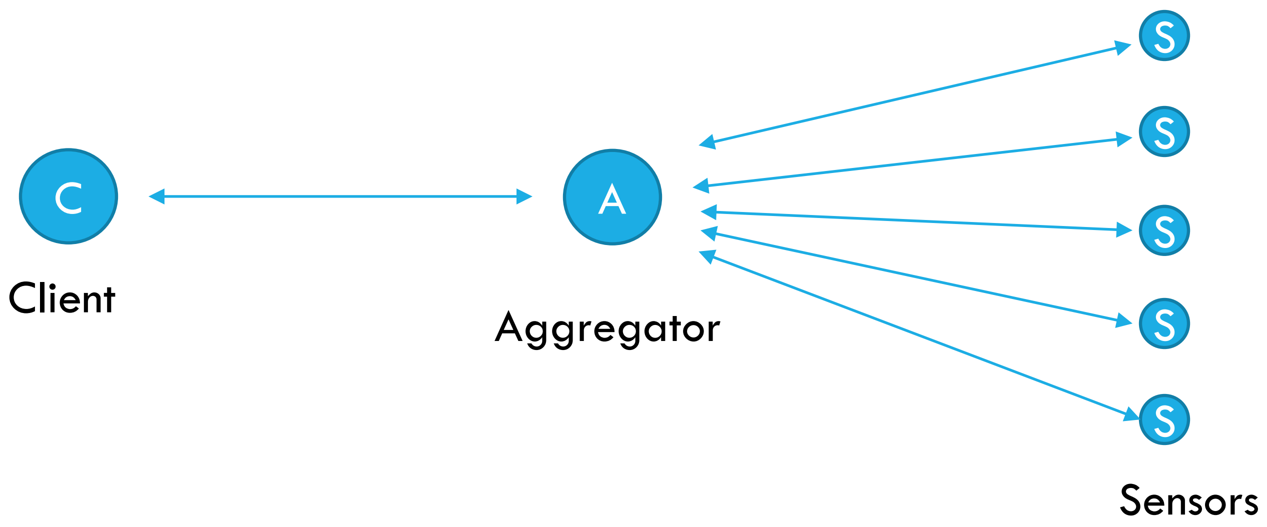
Computer Security Group @ CWI

chenglu.jin@cwi.nl

With Marten van Dijk (CWI), Michael Reiter (Duke University), Sisi Duan (Tsinghua University), and Haibin Zhang (Beijing Institute of Technology)

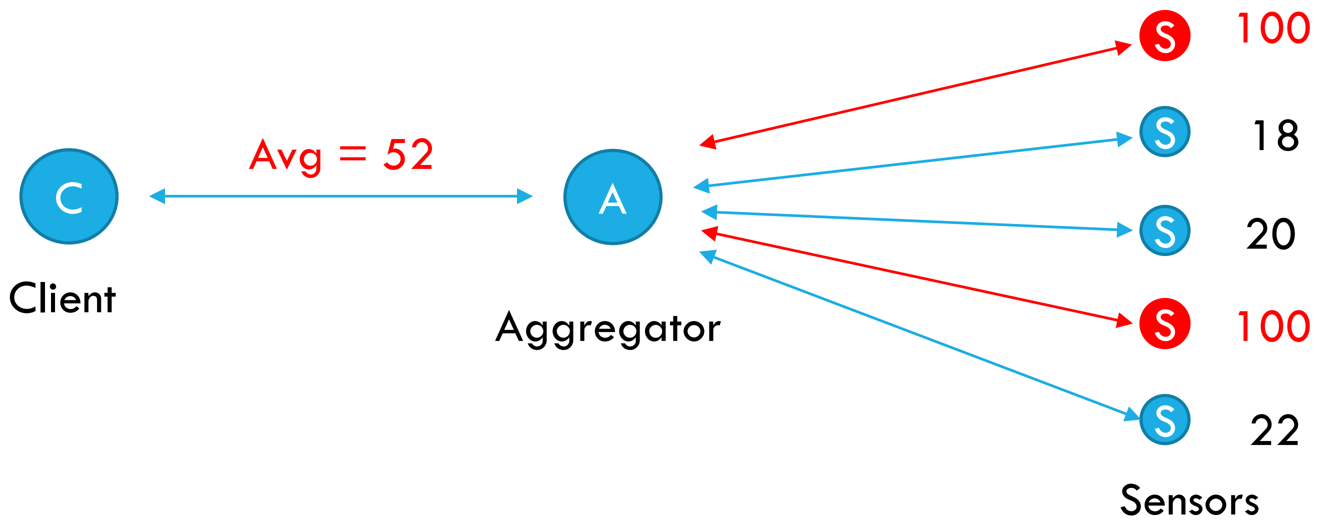
Sensor Data Fusion

- Combine multiple sensor data to produce more dependable and accurate information
- Many applications:
  - Sensor networks
  - Smart metering
- In particular, we are focusing on the applications in sensor-aggregator-client model.



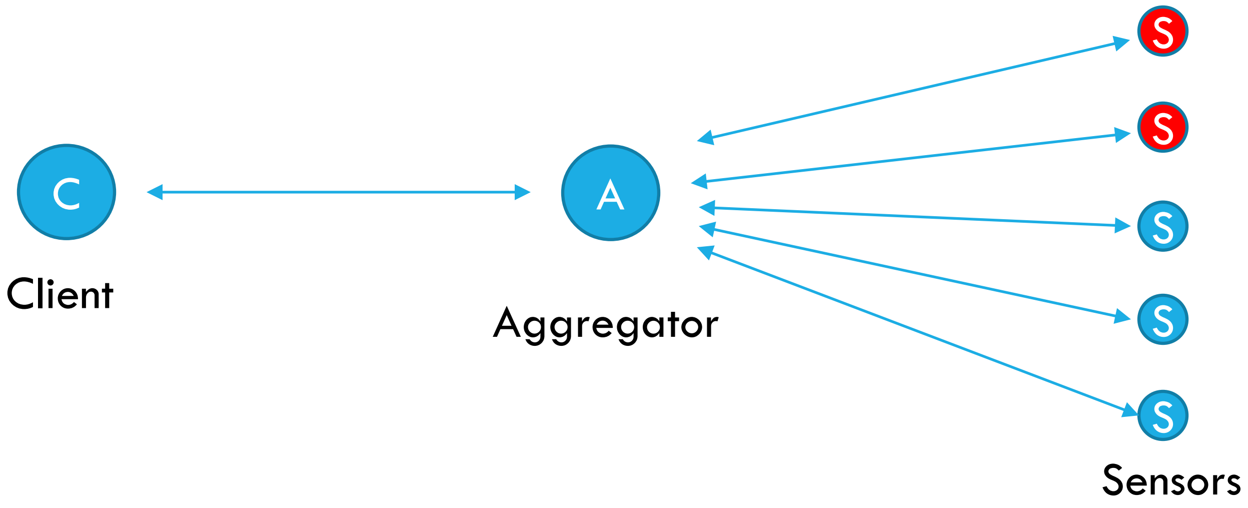
Pollution Attacks

- A small fraction of faulty sensor data can lead to a large error in the aggregated result.
- Existing privacy-preserving works attempt to mitigate the problem by validity proofs of each sensor inputs.
  - Adversaries are still able to change the final output to some extent.



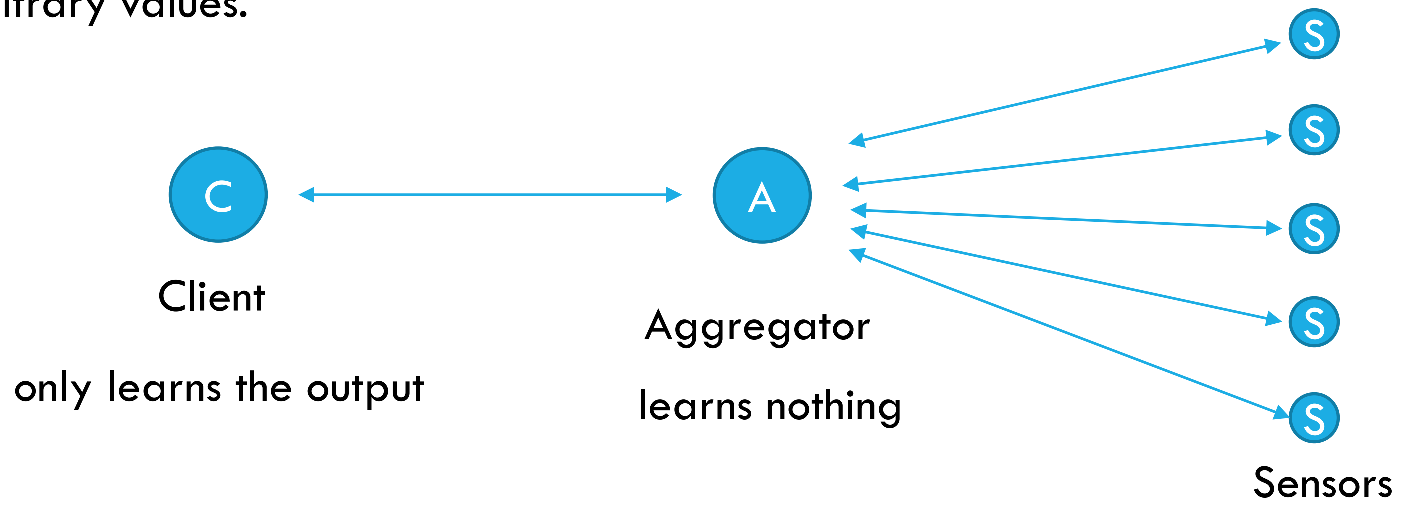
PwoP: Privacy-Preserving and Fault-Tolerant Sensor Fusion

1. Fault tolerant algorithms.
   - Formally defend against pollution attacks given a bound of the fraction of malicious sensors among all the sensors.
   - E.g. Marzullo's algorithm ensures that the result must contain the correct value if at most g out of 2g+1 sensors are malicious



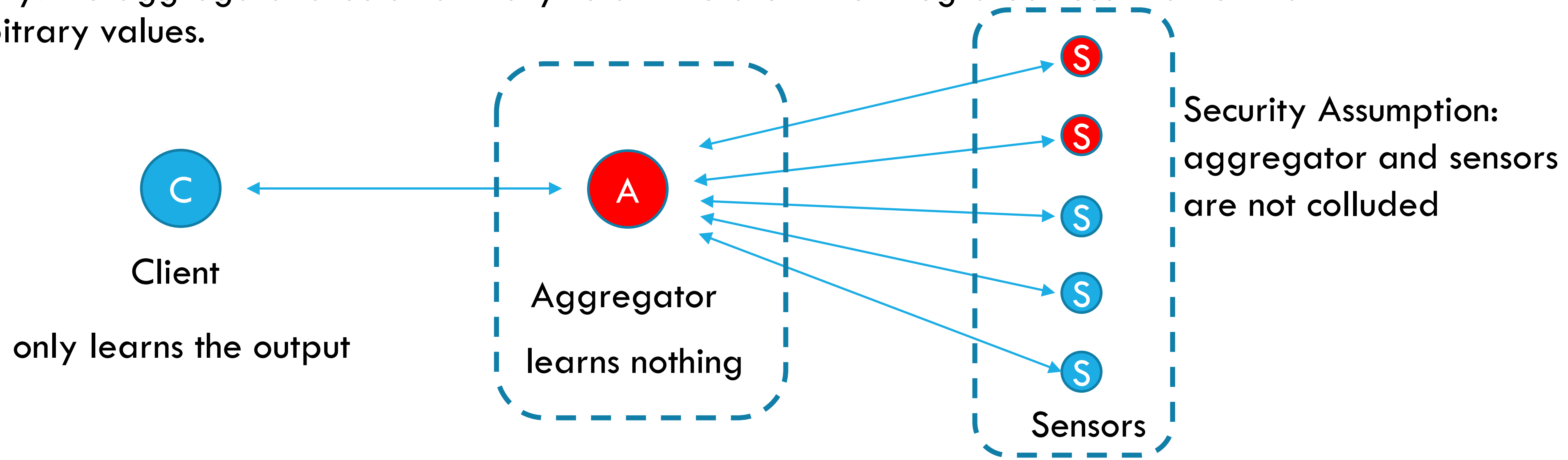
PwoP: Privacy-Preserving and Fault-Tolerant Sensor Fusion

1. Fault tolerant algorithms.
   - Formally defend against pollution attacks given a bound of the fraction of malicious sensors among all the sensors.
   - E.g. Marzullo's algorithm ensures that the result must contain the correct value if at most g out of 2g+1 sensors are malicious
2. Garbled circuits.
   - Privacy: protect the privacy of individual data sources
   - Authenticity: the aggregator should faithfully return the client the aggregated result rather than some arbitrary values.



PwoP: Privacy-Preserving and Fault-Tolerant Sensor Fusion

1. Fault tolerant algorithms.
   - Formally defend against pollution attacks given a bound of the fraction of malicious sensors among all the sensors.
   - E.g. Marzullo's algorithm ensures that the result must contain the correct value if at most g out of 2g+1 sensors are malicious
2. Garbled circuits.
   - Privacy: protect the privacy of individual data sources
   - Authenticity: the aggregator should faithfully return the client the integrated result rather than some arbitrary values.



Garbled Circuits

- Initially designed for secure two-party computation.
- Millionaires' problem

Jeff Bezos

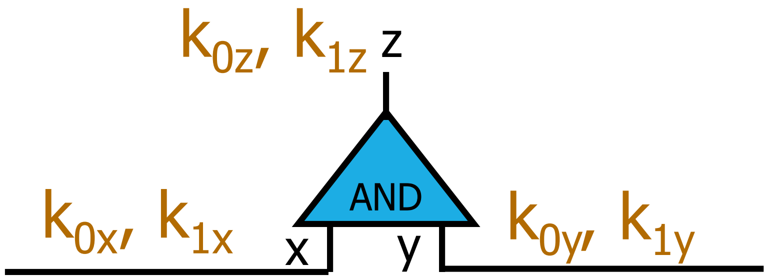
Elon Musk

Who is richer, while keeping privacy?
Without a trusted third party?

Garbled Circuits

Alice (x)

Generator



Bob (y)

Evaluator

$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$

$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

k_{1x}

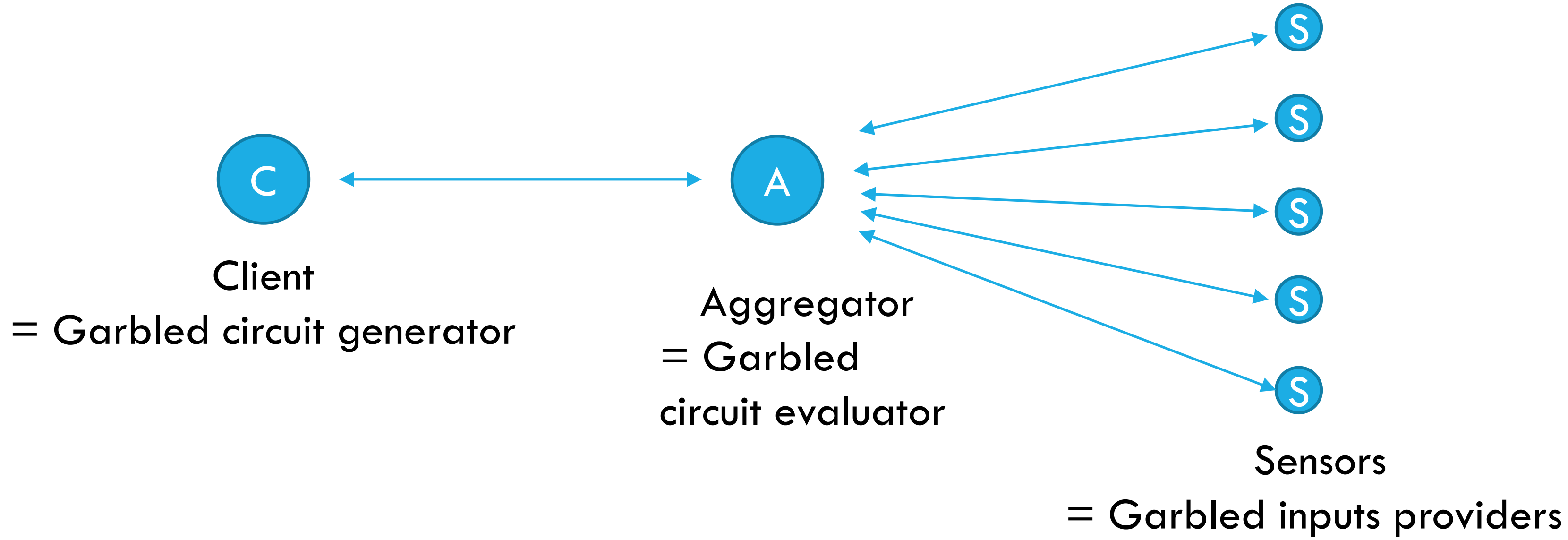
$OT(k_{1y})$

- Reveals nothing more than the output.
- Each party's private input remains secret to the other party.

Open $E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

Apply GC to Our System

- This architecture requires the client to run OT with sensors. Not practical
- We avoid OT by having a pre-shared key between client and each sensor, such that they can derive the same pseudorandom numbers to encode the garbled inputs.
- This key should not be exposed to the aggregator.
  - The reason for non-collusion assumption.

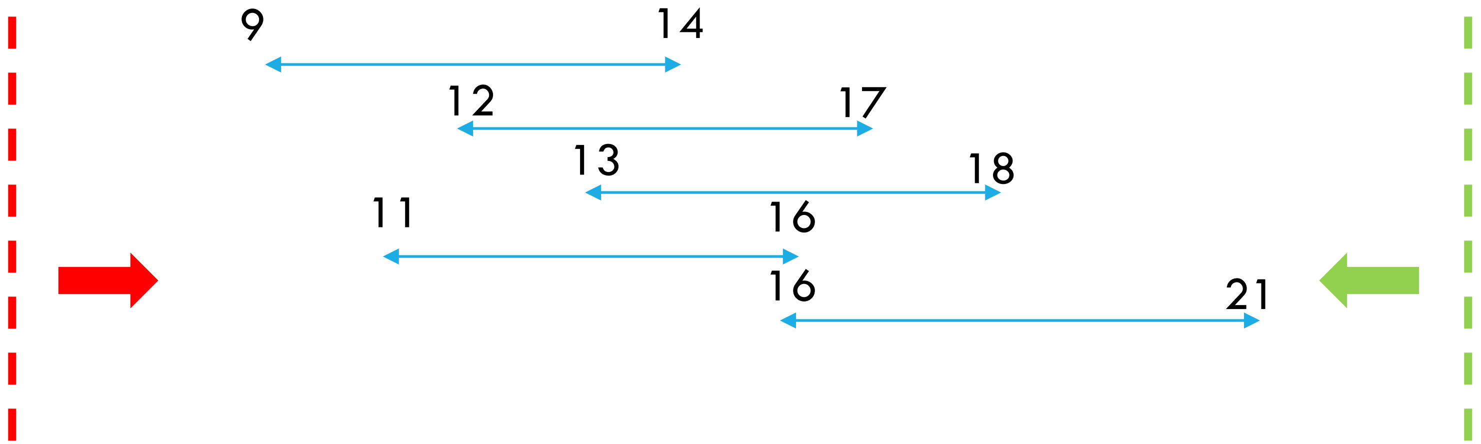


Marzullo's Algorithms

- One of the fault-tolerant sensor averaging algorithms we have studied in our paper.
- It can tolerate g faulty inputs out of $2g+1$ sensor inputs.
- Each sensor input is represented by an interval, which contains the *midpoint* and *accuracy* information.
  - E.g. a sensor input can be (9, 15)

Example

- 5 sensor system, 2 of them can be faulty



Resulting interval

Our Protocol with Semi-Honest Clients

1. Client and sensors generate a shared random coin *r*.
2. Client garbles a fault-tolerant algorithm *f* using *r*, and sends *Gb(f)* to the aggregator.
3. Each sensor generates garbled inputs $En(X_i)$ using *r*, and sends $En(X_i)$ to the aggregator.
4. Aggregator evaluates the garbled circuit with inputs from all the sensors, and outputs a garbled output *Y* to the client.
5. Client runs *DE(Y)* to get *f(X)*

A strong hidden assumption

- The completion of this protocol relies on the fact that all the sensors will provide an input.
- But the sensors are controlled by adversaries.
- The attacker can easily denial-of-service this system after compromising one sensor.

Our Protocol Achieving Liveness

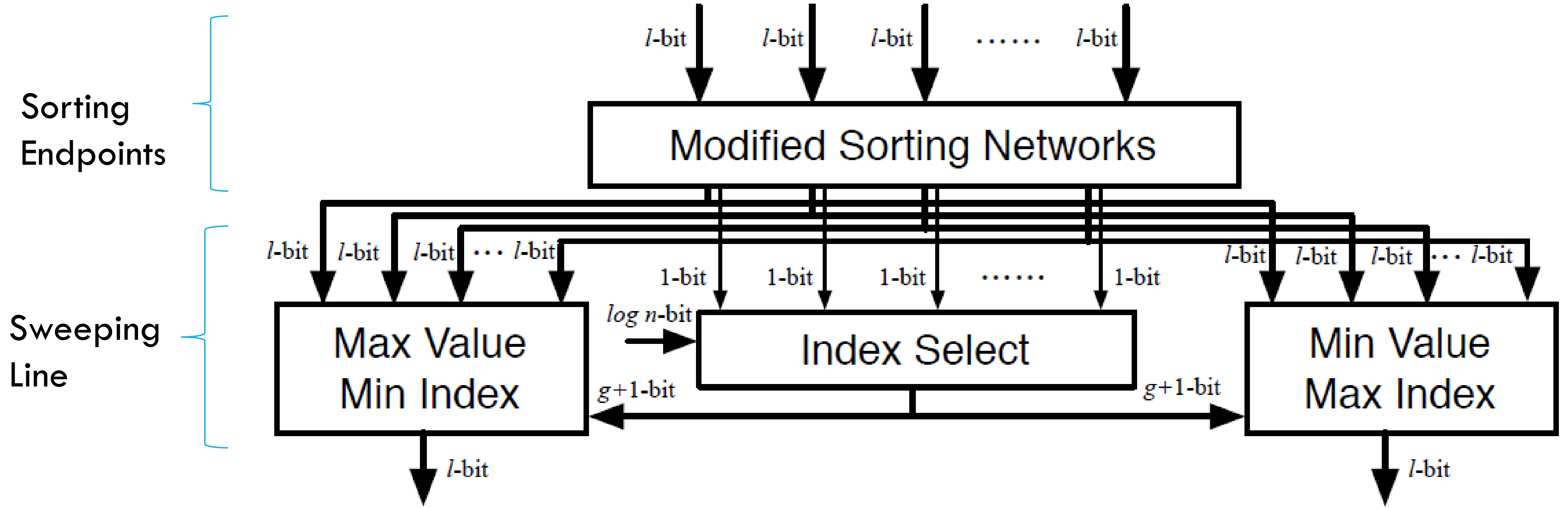
1. Client and sensors generate a shared random coin *r*.
2. Client garbles a fault-tolerant algorithm *f* using *r*, and sends *Gb(f)* to the aggregator.
3. Each sensor generates garbled inputs $En(X_i)$ using *r*, and sends $En(X_i)$ to the aggregator.
4. If aggregator does not receive all the garbled inputs before the times expires, it requests from the client the missing garbled inputs that encode random ranges.
5. Aggregator evaluates the garbled circuit with inputs from all the sensors, and outputs a garbled output *Y* to the client.
6. Client runs $DE(Y)$ to get $f(X)$

Liveness is only possible because our protocol is fault-tolerant.

Implementation

- Optimized the circuit design of Marzullo's algorithm, which saves more than half of the gates comparing with a straightforward design.
- Modified a two party garbled circuit framework /compiler, TinyGarble, to fit our sensor-aggregator-client setting.
- Implemented PwoP in the modified TinyGarble, and evaluated its performance using up to 19 Raspberry Pi Zero W.

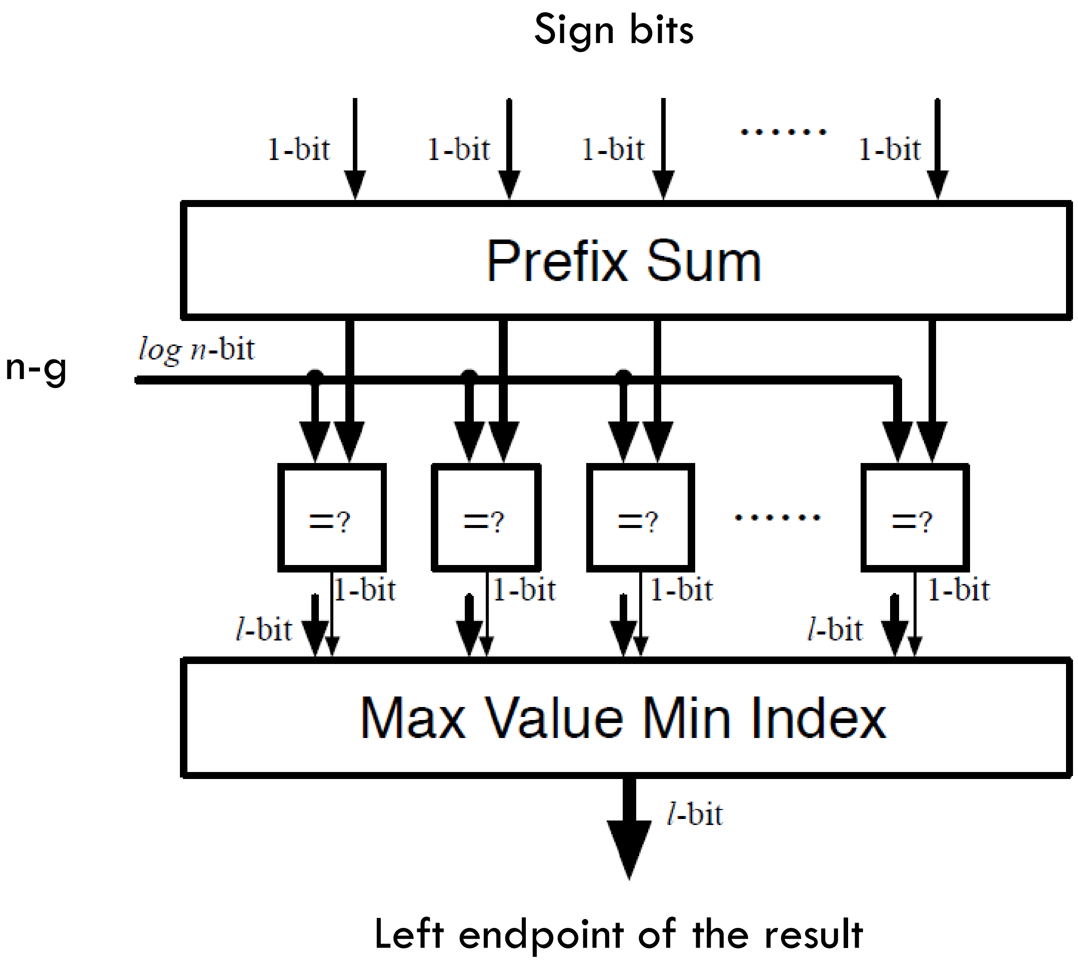
Circuit Design of Marzullo's Algorithm



Why "modified" sorting network?

- Compare the two endpoints provided by each sensor to figure out left and right
- Mark each endpoint with an additional sign bit (1 or -1), indicating it is the left/right one
- Sort all endpoints according to the values of endpoints, and the sign bits need to move together with their associated endpoints.
- Additional checking required by individual algorithms.

Index Select & Max Value Min Index

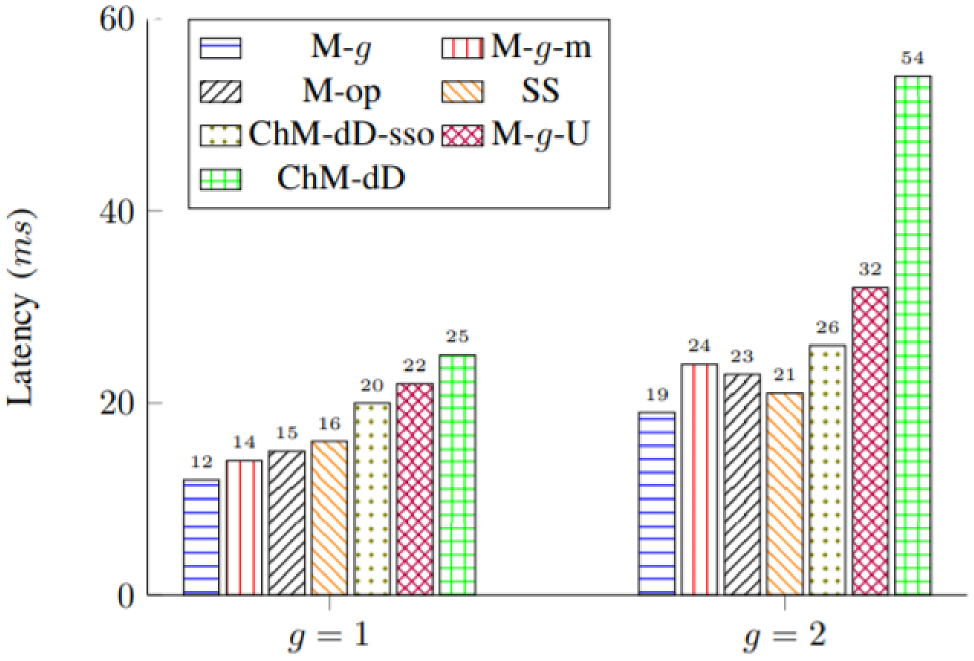


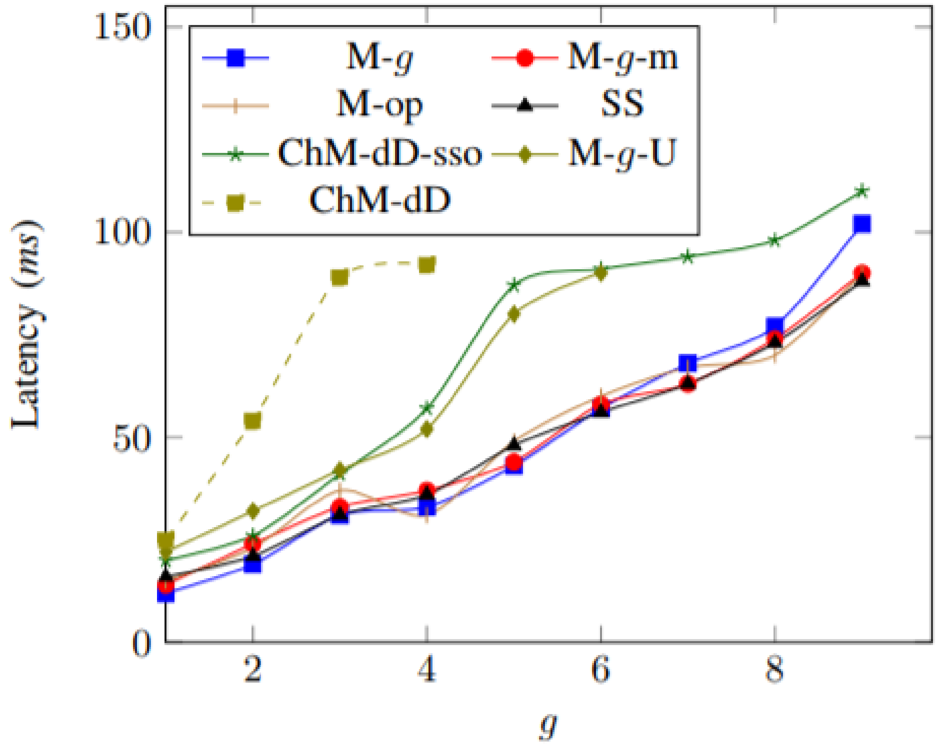
- One prefix sum can be shared for finding both the left index and the right index
- Algorithm specific optimizations can reduce the width of modules and save 75% ~ 84% from a straightforward implementation

More Algorithms

algorithms	type	#sensors	description	complexity	server circuit	sensor time	sensor communication	#rounds
M-g-U	I	$3g+1$	unbounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g	I	$2g+1$	bounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g-m	I	$2g+1$	only reveal midpoint	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-op	I	$2g+1$	"optimistic"	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
SS	I	$2g+1$	Lipschitz condition	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
ChM-dD	dD	$2dg+1$	d-dimension	$O(dn\log(n))$	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$	1 or 2
ChM-dD-sso	dD	$2g+1$	same size & orientation	$O(dn\log(n))$	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$	1 or 2

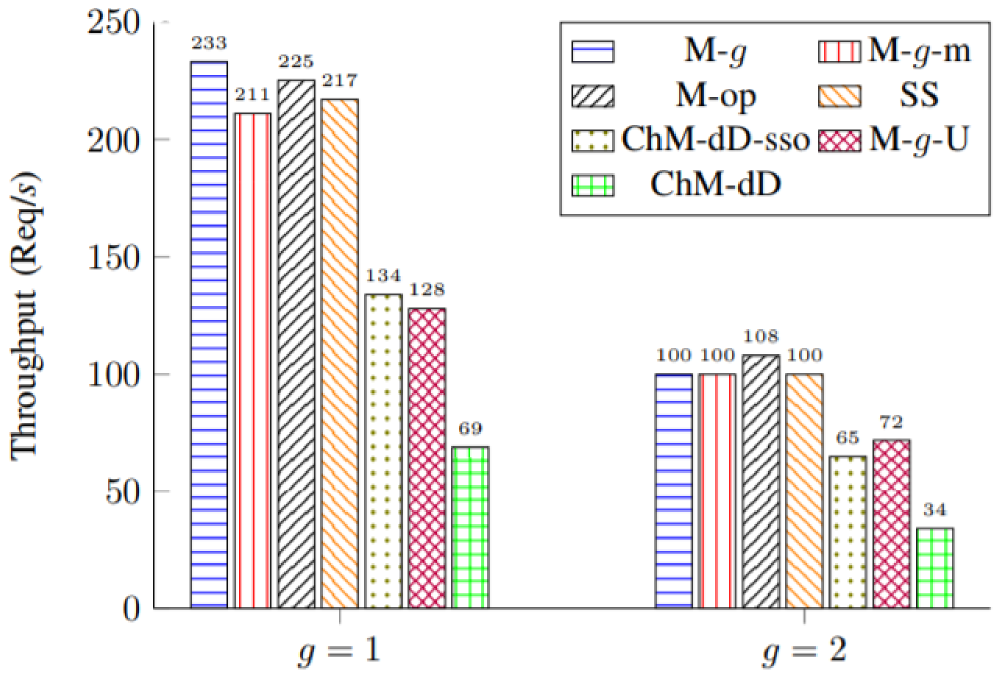
Performance Evaluation (Latency)

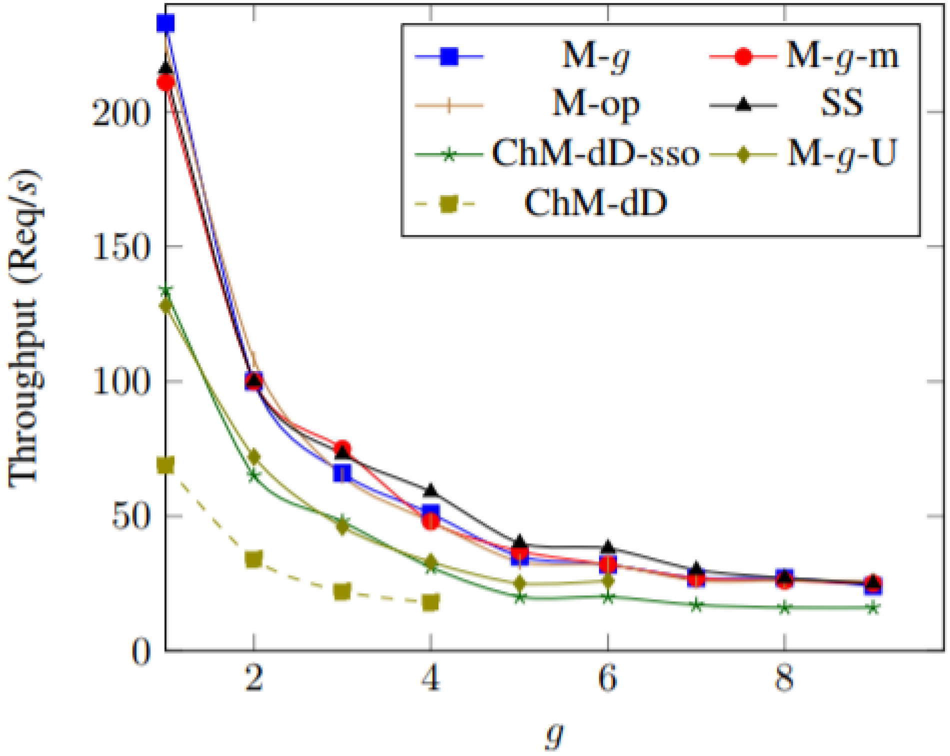




g = # malicious sensors

Performance Evaluation (Throughput)





g = # malicious sensors

Summary

- We formalized the problem of server-aided, privacy-preserving, fault-tolerant sensor fusion.
- We provided an efficient framework supporting a variety of fault-tolerant sensor fusion algorithms.
- We built a practical system for server-aided multi-sensor fusion using Raspberry Pi Zero W and WiFi communication.