PG: Byzantine Fault-Tolerant and Privacy-Preserving Sensor Fusion with Guaranteed Output Delivery

Chenglu Jin*[1], Chao Yin*[2,1], Marten van Dijk[1,2], Sisi Duan[3], Fabio Massacci[2], Michael K. Reiter[4], Haibin Zhang[5]

Email: chenglu.jin@cwi.nl

* Shared first-authorship

[1] Centrum Wiskunde & Informatica (CWI Amsterdam), [2]Vrije University Amsterdam, [3]Tsinghua University, [4]Duke University, [5]Yangtze Delta Region Institute of Tsinghua University, Zhejiang

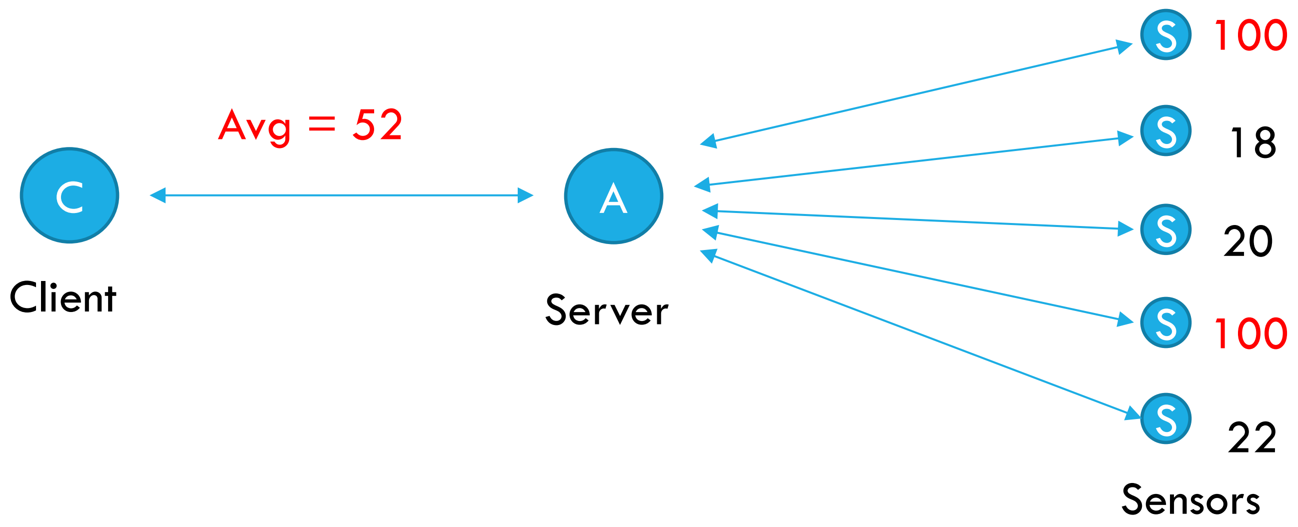
Published at ACM Conference on Computer and Communications Security (CCS) 2024

Outline

- Technical overview
- Background
- P0, privacy-preserving and fault-tolerant
- P1, achieving guaranteed output delivery (GOD) in the crash failure model
- P2, achieving GOD in the Byzantine failure model
- P3, realizing privacy against malicious servers
- Implementation Optimizations
- Experimental evaluation

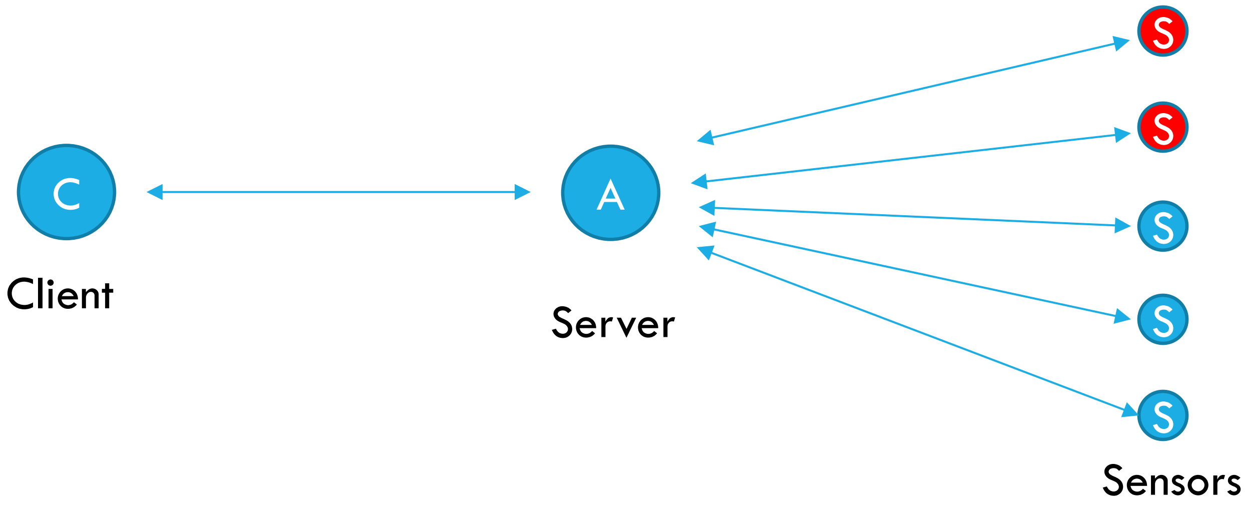
Sensor Data Fusion

- Combine multiple sensor data to produce more dependable and accurate information. E.g., sensor networks, smart metering.
- In particular, we are focusing on the client-server-sensor model.
- Pollution attack: a small fraction of faulty sensor data can lead to a large error in the aggregated result.



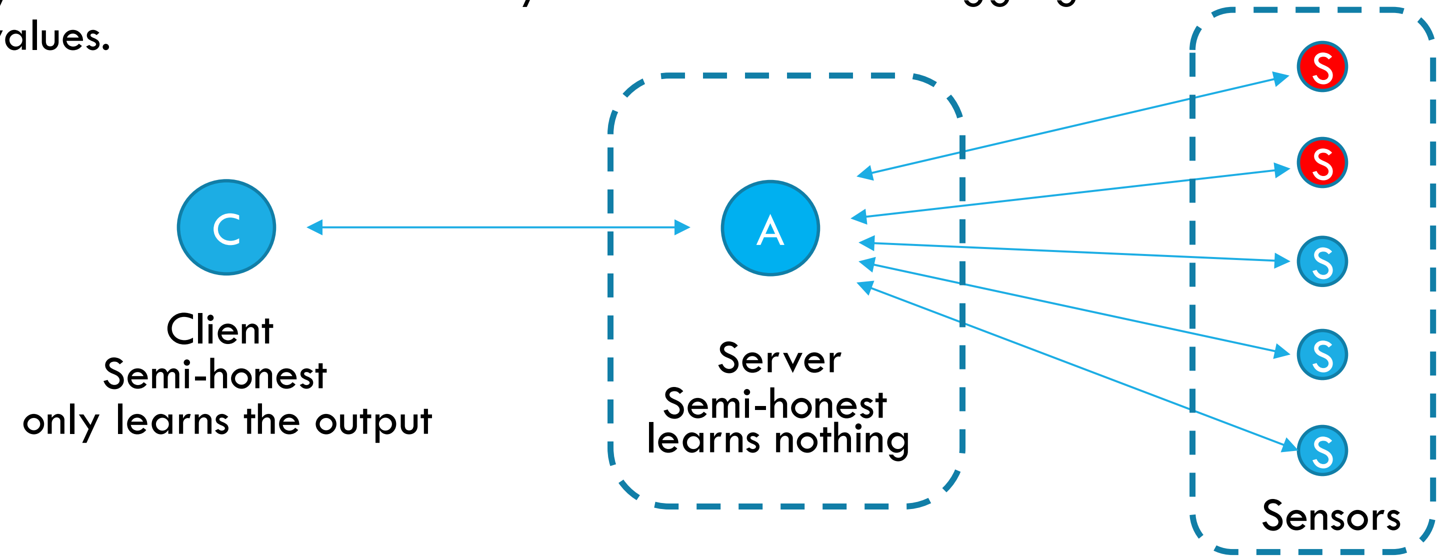
PG: Privacy-Preserving and Fault-Tolerant Sensor Fusion

1. Fault tolerant algorithms (FTA).
   - 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



PG: Privacy-Preserving and Fault-Tolerant Sensor Fusion

1. Fault tolerant algorithms (FTA).
   - 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 (GC).
   - Privacy: protect the privacy of individual sensor inputs
   - Authenticity: the server should faithfully return the client the aggregated result rather than some arbitrary values.

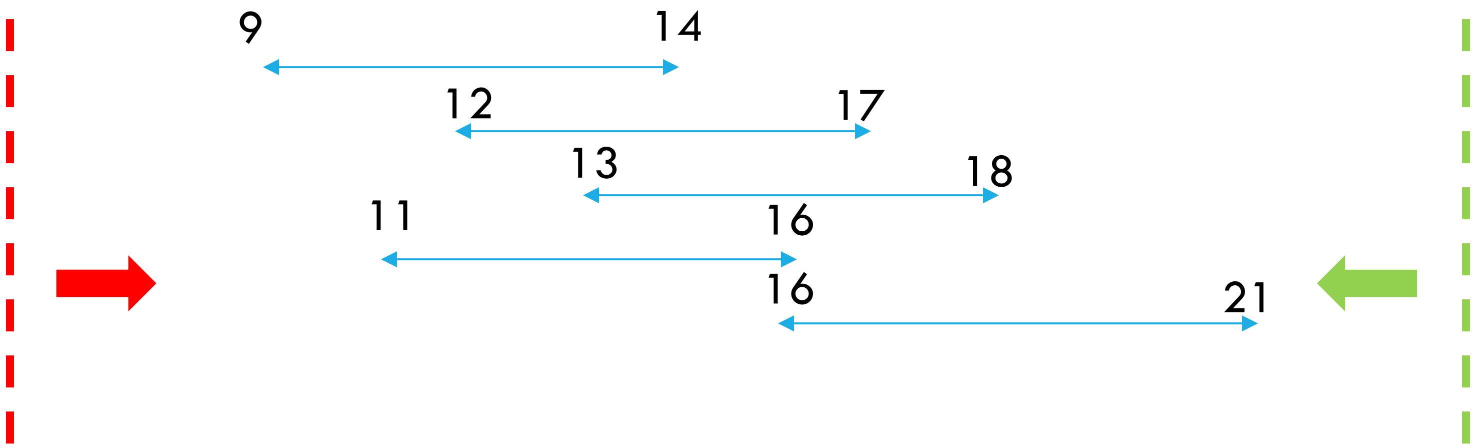


Malicious sensors can collude

Security Assumption: server and sensors do not collude

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, 14)



Resulting interval

Garbled Circuits

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

Jeff Bezos

Elon Musk

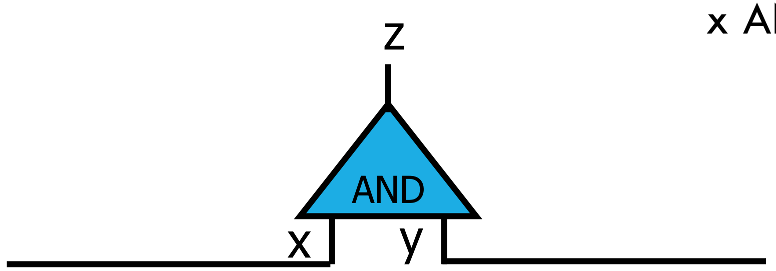
Who is richer, while keeping privacy?
Without a trusted third party?

Garbled Circuits

Alice, x

Generator

x AND y = ?



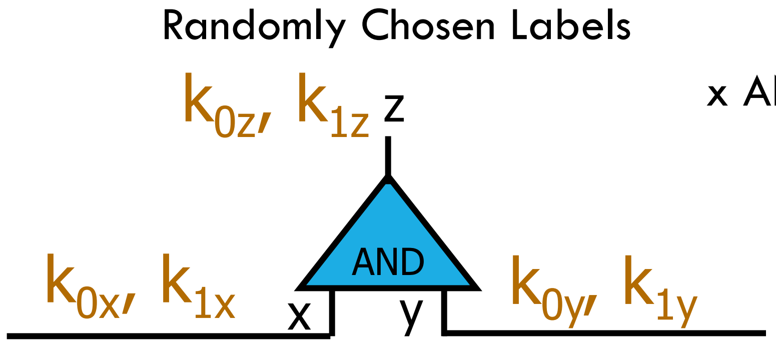
Bob, y

Evaluator

Garbled Circuits

Alice, x

Generator



x AND y = ?

Bob, y

Evaluator

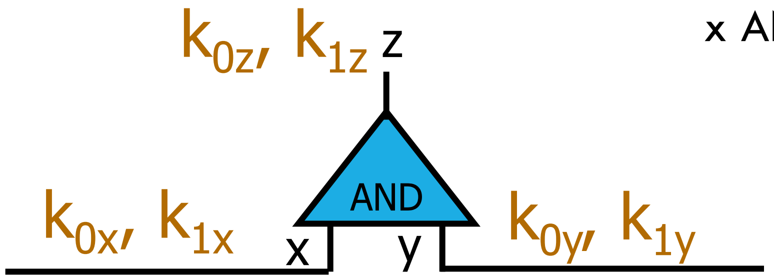
Garbled Circuits

Randomly Chosen Labels

x AND y = ?

Alice, x

Generator



Bob, y

Evaluator

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

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

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

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

Garbled Circuits

Randomly Chosen Labels

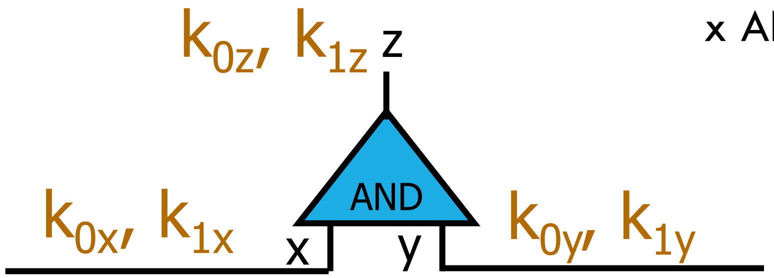
x AND y = ?

Alice, x

Generator

Bob, y

Evaluator



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

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

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

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

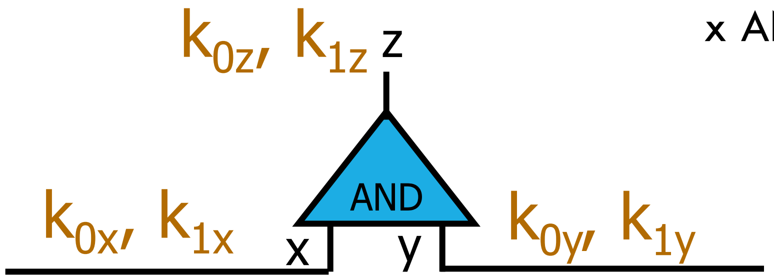
Garbled Circuits

Randomly Chosen Labels

x AND y = ?

Alice, x

Generator



Bob, y

Evaluator

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

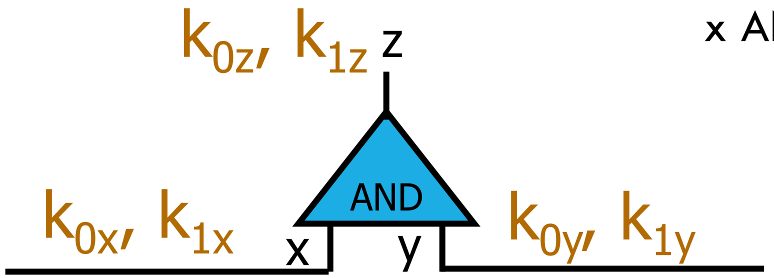
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

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

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

Garbled Circuits

Randomly Chosen Labels



x AND y = ?

Alice, x

Generator

Bob, y

Evaluator

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

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

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

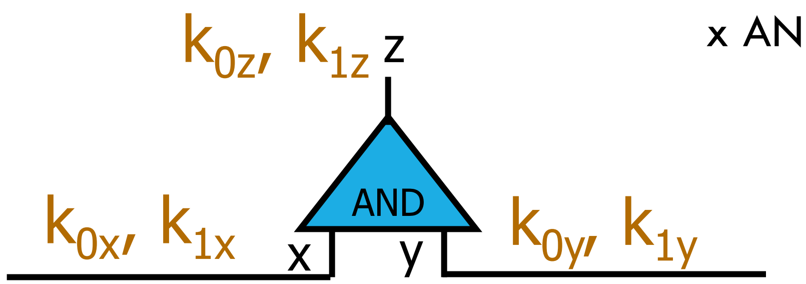
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

k_{1x}

CWI

Garbled Circuits

Randomly Chosen Labels



x AND y = ?

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}

CWI

Garbled Circuits

Randomly Chosen Labels

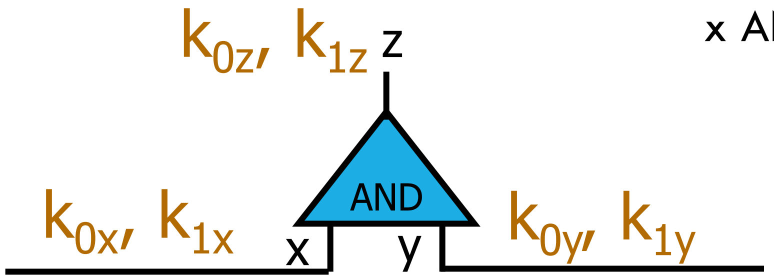
x AND y = ?

Alice, x

Generator

Bob, y

Evaluator



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

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

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

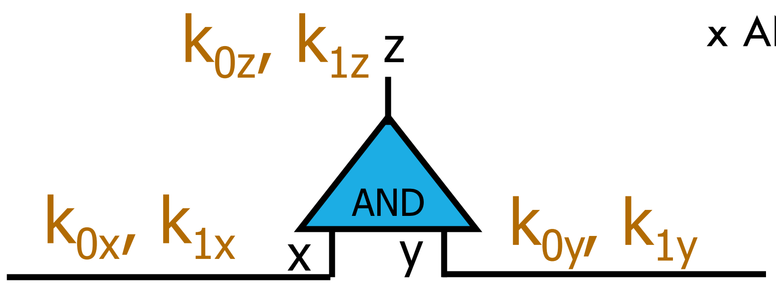
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

k_{1x}

Oblivious Transfer(k_{1y})

Garbled Circuits

Randomly Chosen Labels



x AND y = ?

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}

Oblivious Transfer(k_{1y})

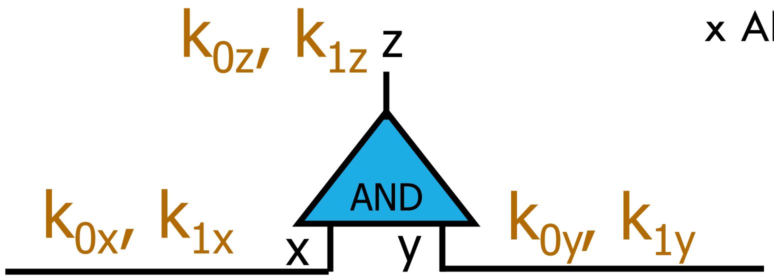
Garbled Circuits

Randomly Chosen Labels

x AND y = ?

Alice, x

Generator



Bob, y

Evaluator

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

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

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

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

Only one ciphertext will be decrypted

k_{1x}

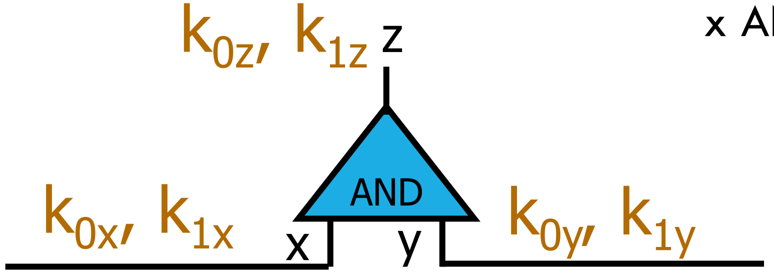
Oblivious Transfer(k_{1y})

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

Garbled Circuits

Randomly Chosen Labels

x AND y = ?



Alice, x

Generator

Bob, y

Evaluator

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

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

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

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

Only one ciphertext will be decrypted

k_{1x}

Oblivious Transfer(k_{1y})

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

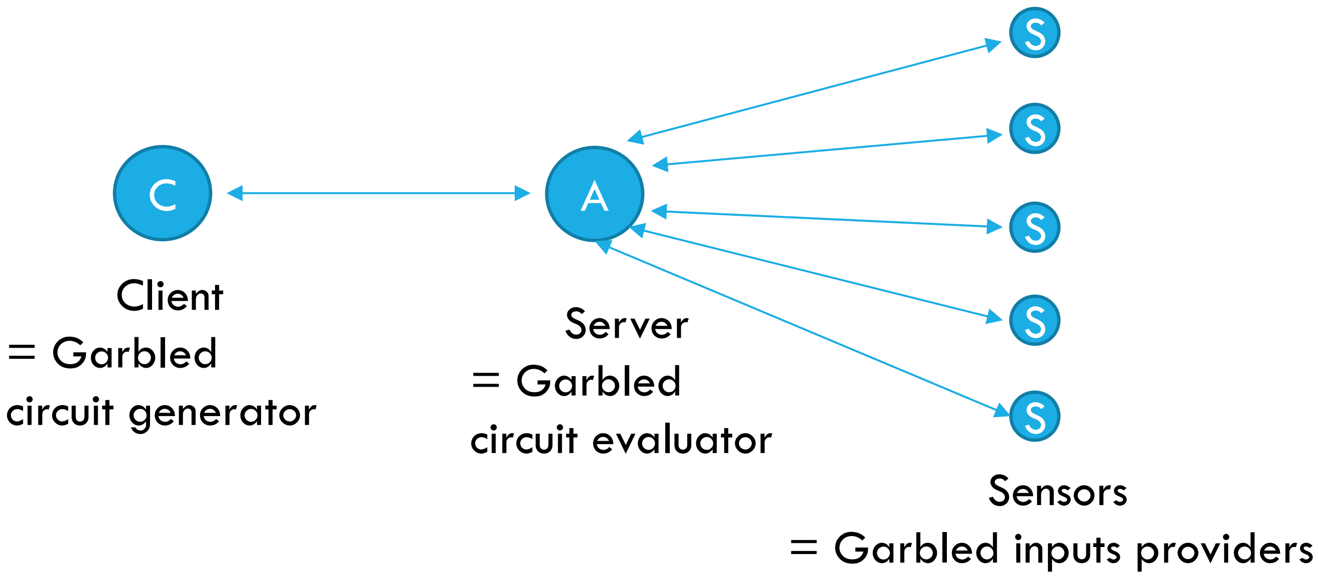
- Reveals nothing more than the output, due to the random label encoding.
- Each party's private input remains secret to the other party.

P0: Apply GC and FTA to Our System

- We use a pre-shared secret key between the client and each sensor to derive the same randomness needed to garble the circuit or the inputs
- This key should not be exposed to the server.

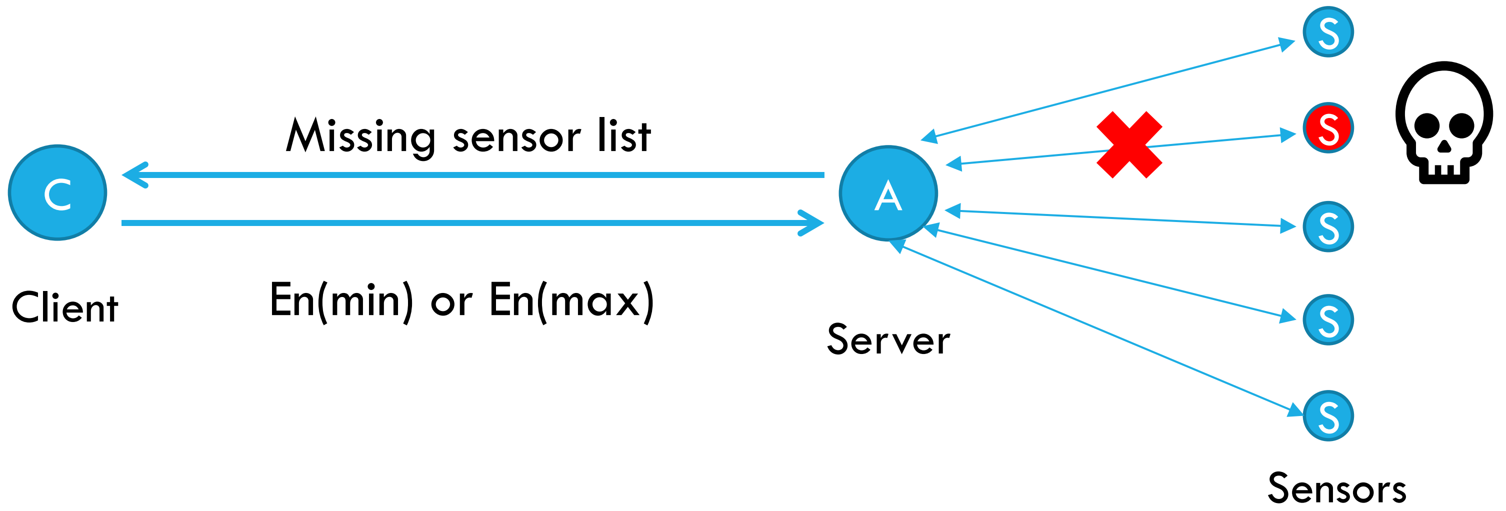
1. The client garbles a fault-tolerant algorithm *f()* that performs the sensor fusion and sends the garbled circuit *Gb(f)* to the server.
2. Server fetches garbled inputs $En(X_i)$ from the sensors
3. Server evaluates the garbled circuit
4. Garbled output *Y* is sent to Client
5. Client decodes De(Y) to get f(X)

- Input Privacy
- Output integrity
- Tolerate incorrect sensor inputs



P1: Achieving GOD in the Crash Failure Model

- The completion of P0 protocol requires all the sensors to provide an input.
- Easy DoS attack by compromising just one sensor and not sending anything.
- One more round of interaction: if the server does not receive all the garbled inputs before a timer expires, it requests from the client for the missing sensor inputs that will encode the minimum or maximum.
- The missing inputs become faulty inputs that will be tolerated by FTA
- GOD is achievable because our protocol is fault-tolerant.

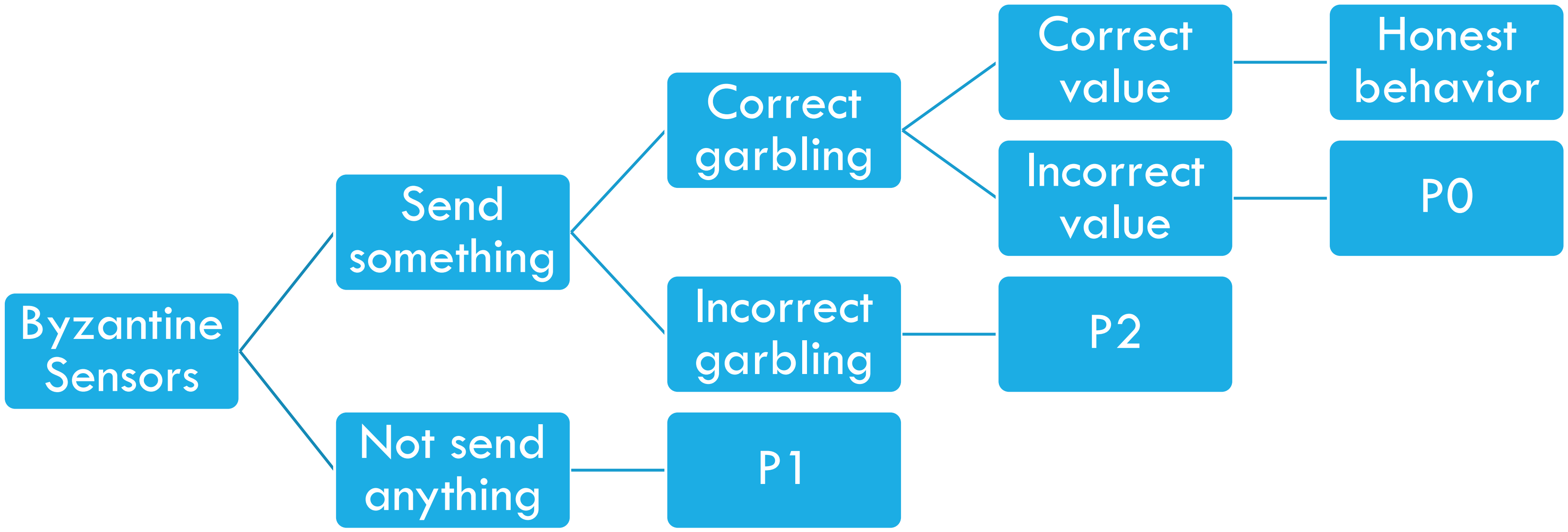


P2: Achieving GOD in the Byzantine Failure Model

- Byzantine failure model means the malicious sensors can do anything they want
- Sensors may send ill-formed inputs (not correctly garbled inputs)
- Easy DoS attack by compromising one sensor and always sending ill-formed inputs. The aggregated result is not decodable by the client. It cannot be caught by the client, due to the privacy guarantee of GC.
- Same protocol interaction as P1, but adding checking gates (encrypted truth tables) besides the functional circuit to detect ill-formed inputs:
  - All-zero-strings, instead of output labels, are encrypted by valid input labels pairs
  - If all inputs from a sensor pass the check, use them in the functional circuit; otherwise request valid labels that encode minimum/maximum from the client.
  - For N-bit inputs, we need to add additional N/2 checking gates, regardless of the functional circuit

Recap: Byzantine Malicious Sensors

- What can a (group of) Byzantine malicious sensor(s) do?



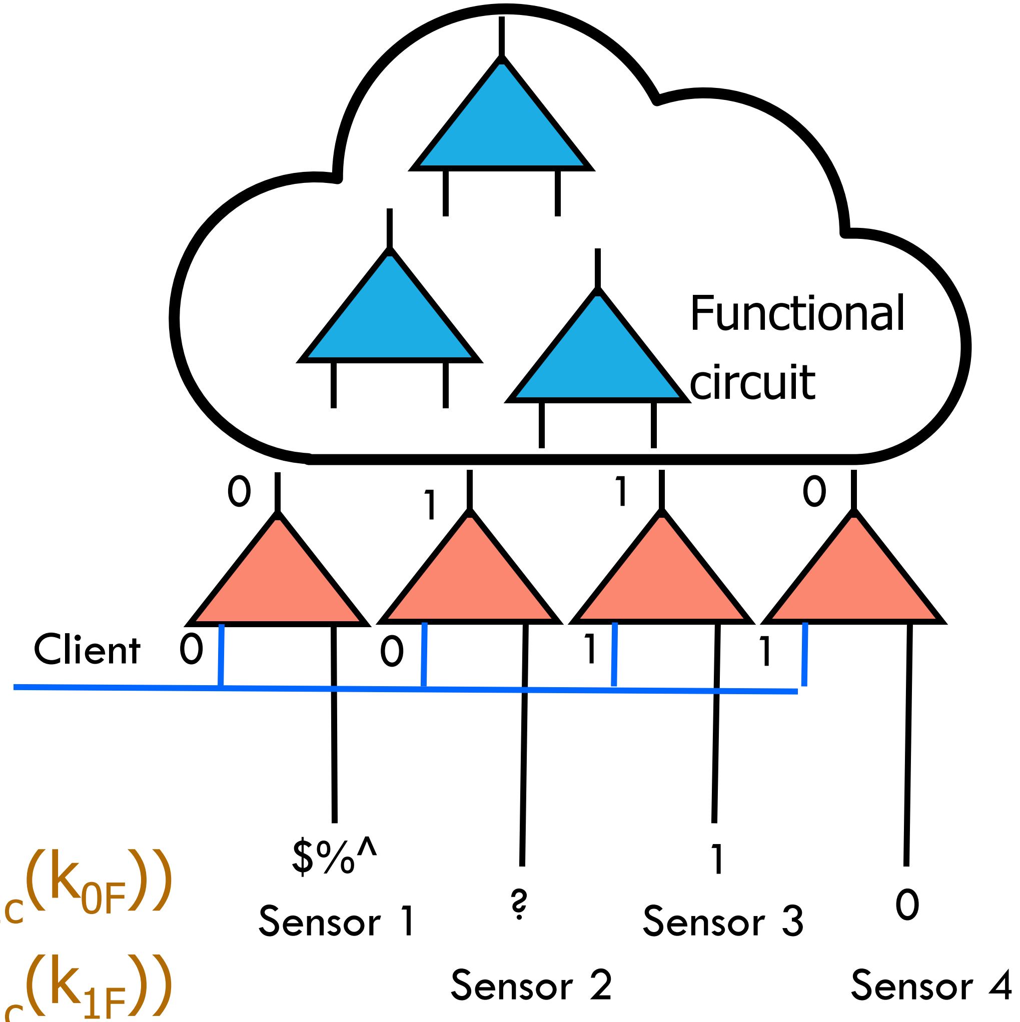
As long as the total number of malicious sensors does not exceed the threshold of the fault-tolerant algorithm, all combinations of possible colluding malicious behaviors will be tolerated

From Semi-Honest Server to Malicious Server

- What a malicious server can do without being detected by the client?
  - A detectable malicious behavior (e.g., not responding) can be solved by switching to a different server
- Traditional garbled circuit is secure against a malicious evaluator
- Malicious server controls the list of sensors that did not provide valid sensor labels, and this list will be sent to the client for requesting the missing input labels
- A malicious server may manipulate the list and request a set of valid inputs of honest sensors from the client
  - It will obtain two sets of valid input labels of the same set of wires, allowing the malicious server to evaluate the same circuit with multiple different inputs.
  - Tamper with the final result by flipping some of the input wires (integrity violation)
  - Differential analysis on the computation results may leak honest sensor's input values (privacy violation)
  - With FreeXOR optimization, leaking one pair of valid labels on the same wire exposes all of the valid labels on all of the wires in the same circuit.

P3: Secure against a Malicious Server

- Filter gate (encrypted truth table): if the server "**claims**" some sensor input labels are missing/ill-formed, the client sends labels to help decrypt one row of the truth table to get valid output labels that encodes the min/max; otherwise the client sends labels to forward the sematic values of sensor inputs to the output labels.
- Implicitly transfer the max/min values of the functional circuit inputs via the filter gate truth tables
- Guarantee: at most one label per wire can be revealed to the server
- Still need to assume that the malicious server and the malicious sensors do not collude

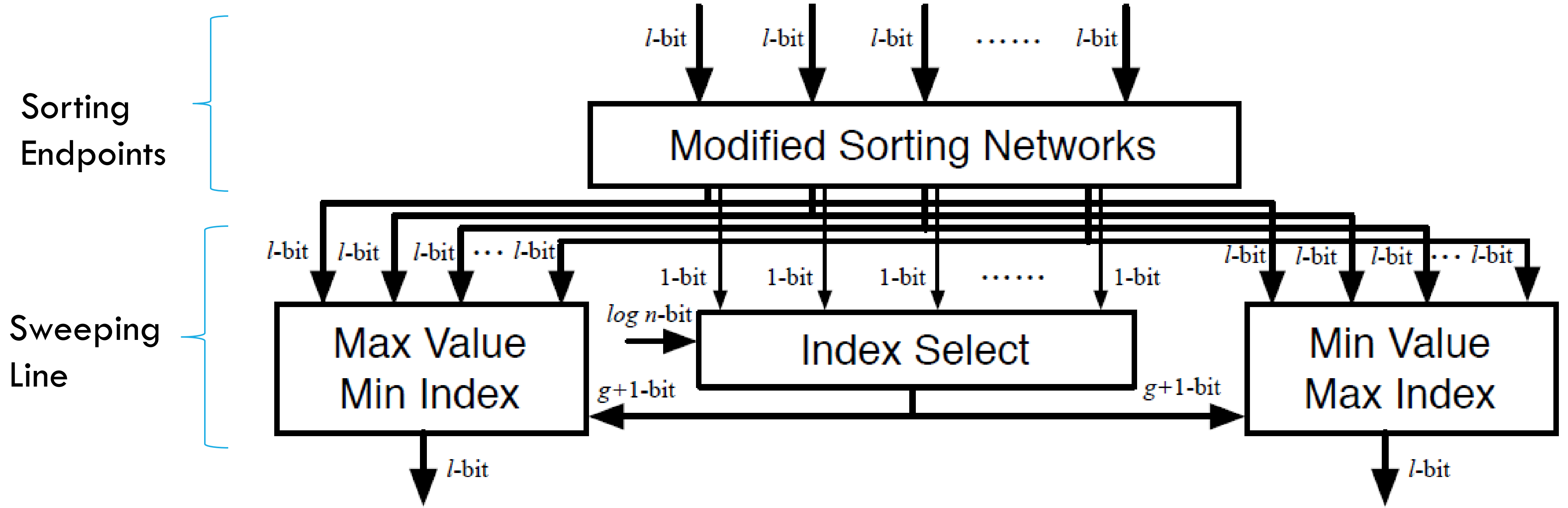


$$E_{k_{0s}}(E_{k_{1c}}(k_{0F}))$$

$$E_{k_{1s}}(E_{k_{1c}}(k_{1F}))$$

$$E_{k_{0c}}(k_{0F}))$$

Circuit Design of Marzullo's Algorithm

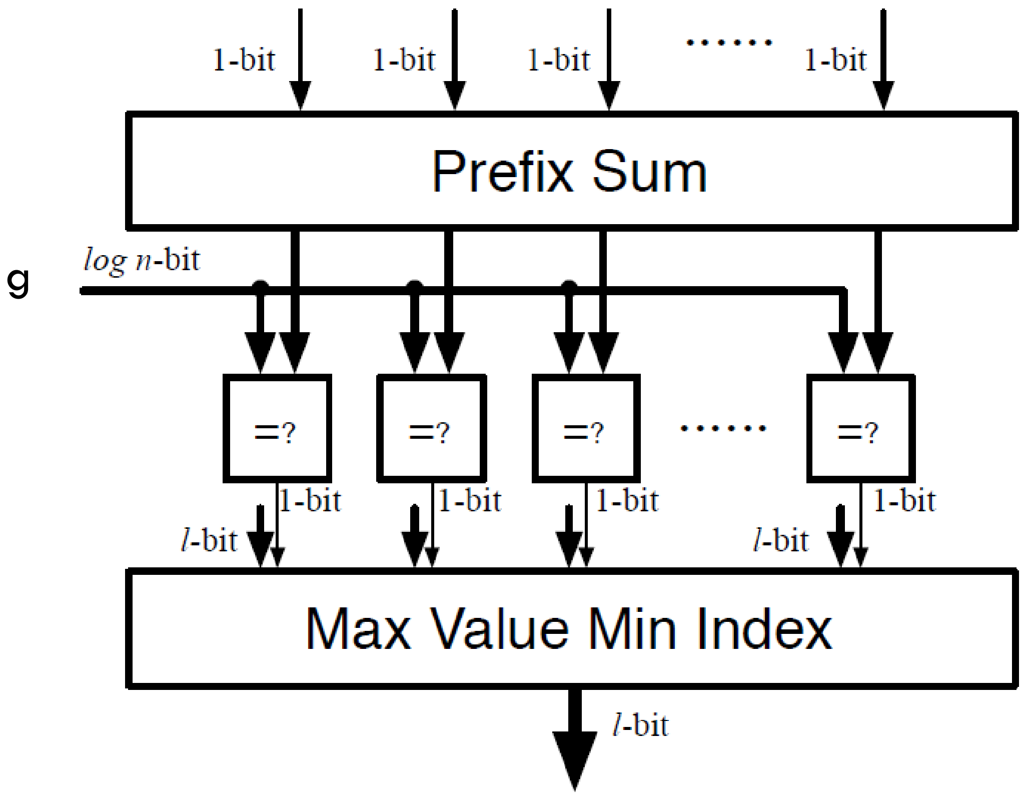


Why "modified" sorting network?

- Compare the two endpoints provided by each sensor to figure out left and right
- Mark each endpoint with an additional sign bit (1 or 0), 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

Sign bits



Left endpoint of the result

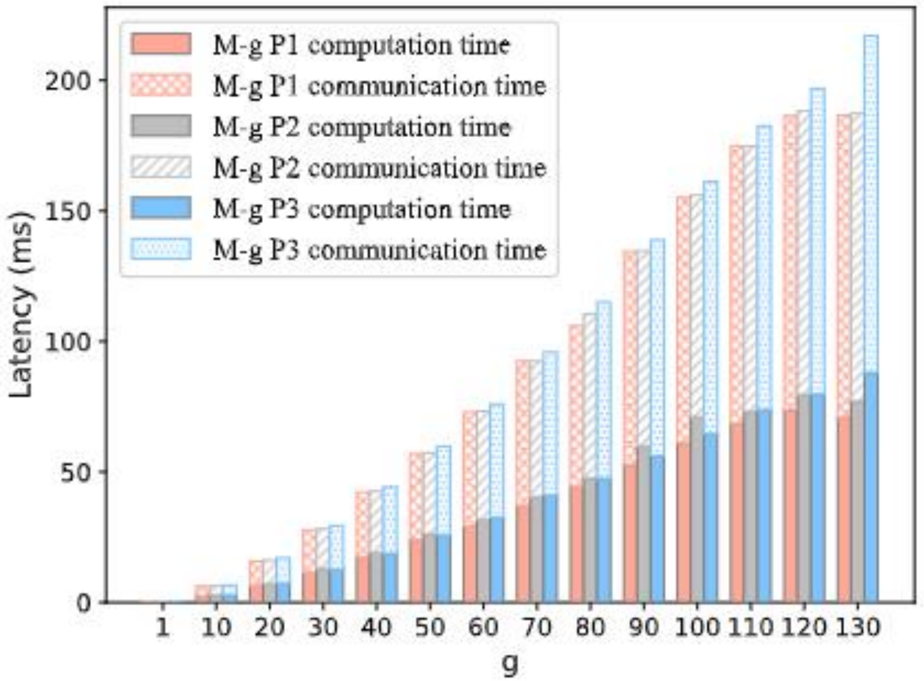
- One prefix sum can be shared for finding both the left index and the right index
- Algorithm specific optimization can reduce the width of modules and save 75% ~ 84% from a straightforward implementation

More Algorithms

algorithms	#sensors	description	complexity	server circuit	sensor time	sensor communication	#rounds
M-g-U	$3g+1$	unbounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g	$2g+1$	bounded accuracy	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-g-m	$2g+1$	only reveal midpoint	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
M-op	$2g+1$	"optimistic"	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2
SS	$2g+1$	Lipschitz condition	$O(n\log(n))$	$O(ln\log^2(n))$	$O(l)$	$O(lk)$	1 or 2

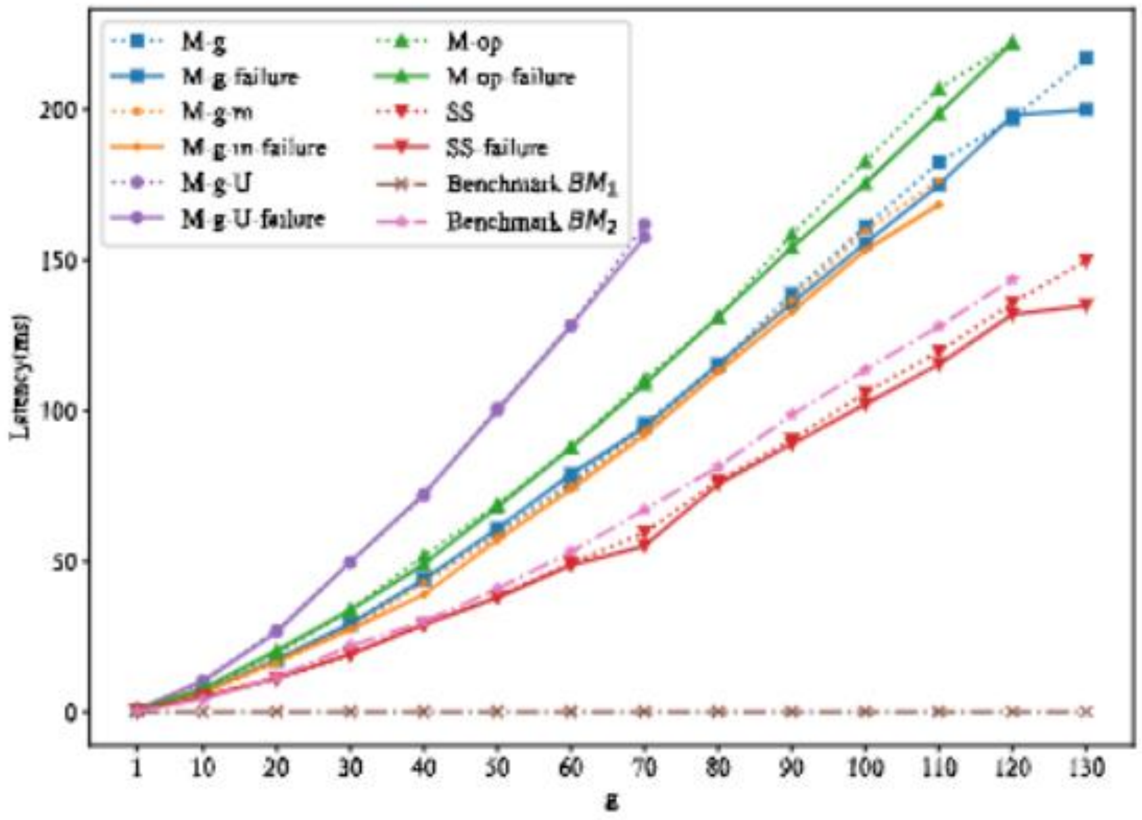
Cloud Evaluation

- Modified a two party garbled circuit framework, TinyGarble, to fit our sensor-server-client setting
- Simulated a sensor network on AWS cloud. Each sensor/server/client is one AWS node. Used up to 261 sensor nodes
- System latency scales well with the number of sensors



Failure-free vs Failure Performance of P3

- Latency is smaller in the failure scenario because the server skips the communication with the failed sensors



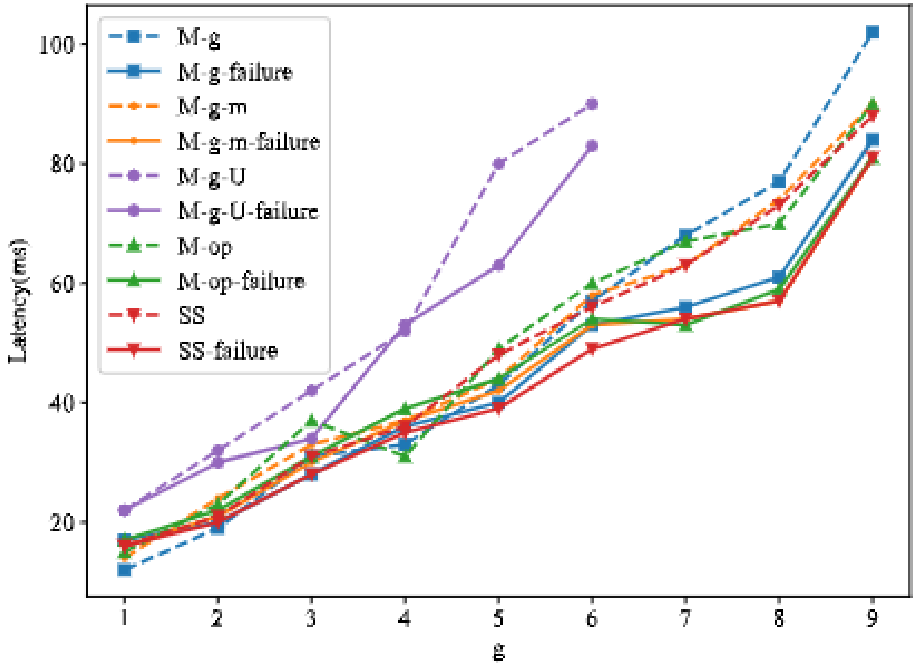
CWI

Cyber-Physical System Implementation

- Implemented PG.P1 in a cyber-physical system setting, and evaluated its performance using up to 19 Raspberry Pi Zero W.
- The client is a commercial laptop, and the server is a desktop.
- The server and all the sensors are communicated over WiFi via a router.
- The client and the server are connected by an Ethernet cable.

CPS Evaluation

- The performance is scalable.
- Not as fast as the cloud due to the local WiFi connection and a computationally constrained devices emulating the sensors



Summary

- Design an efficient and scalable framework for privacy-preserving and Byzantine fault-tolerant sensor fusion. It fits the resource constrained sensors.
- Develop new techniques to achieve guaranteed output delivery in GC when a fraction of sensors are Byzantine malicious.
- Extend our system to be secure against a malicious server.
- Optimize the circuits designs realizing the fault-tolerant sensor fusion algorithms.
- Evaluate the performance of our system on AWS cloud with up to 261 sensors and a cyber-physical system with up to 19 sensors.
- Source code: https://figshare.com/articles/software/PG_source_code/25669026

Thank you! Questions?

Email: chenglu.jin@cwi.nl