CS 350S: Privacy-Preserving Systems

Private aggregate statistics

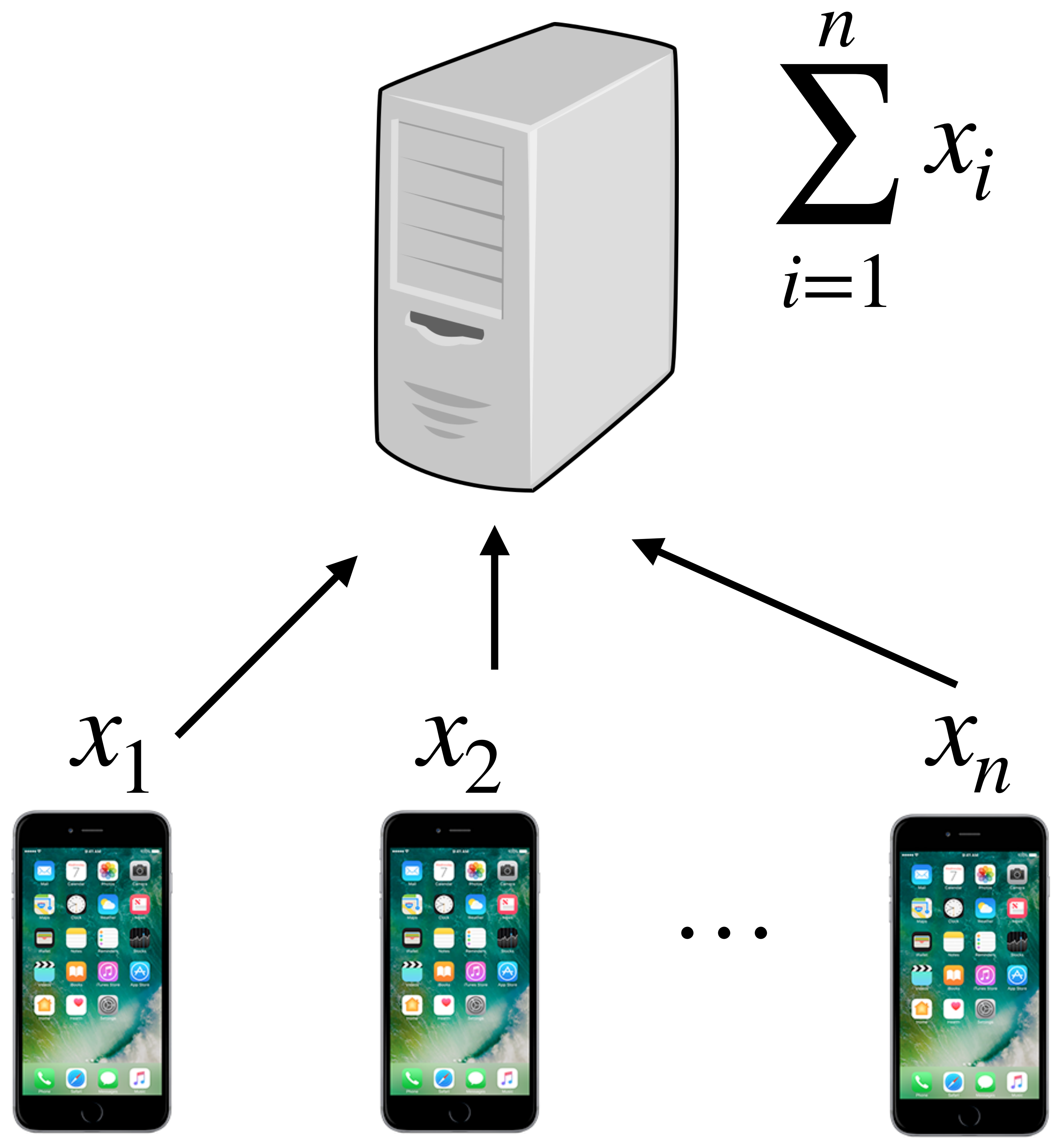
Outline

1. **Prio design**
2. Tim Geoghegan and Chris Patton

Private aggregation

Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

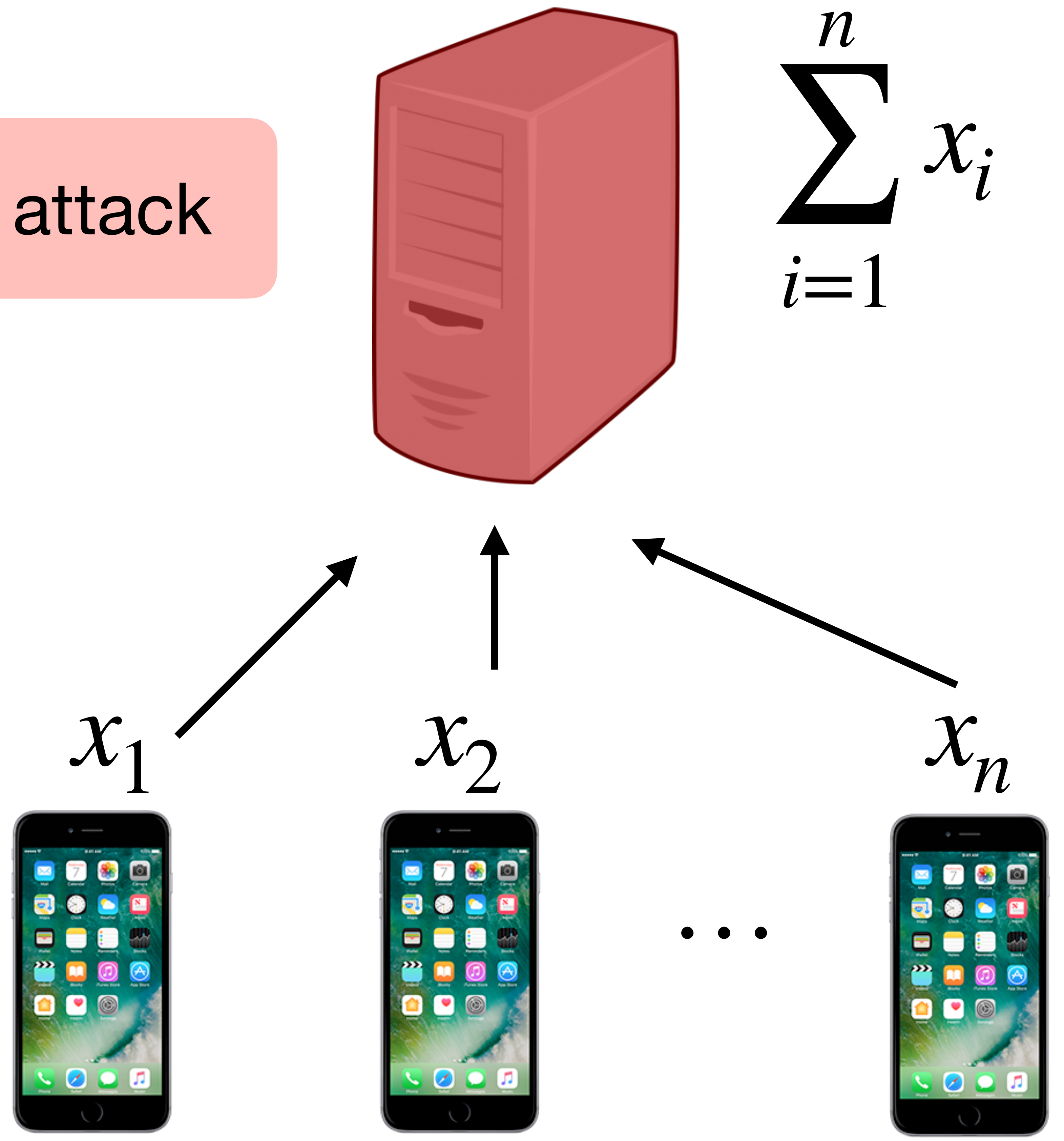


Private aggregation

Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

Server is central point of attack



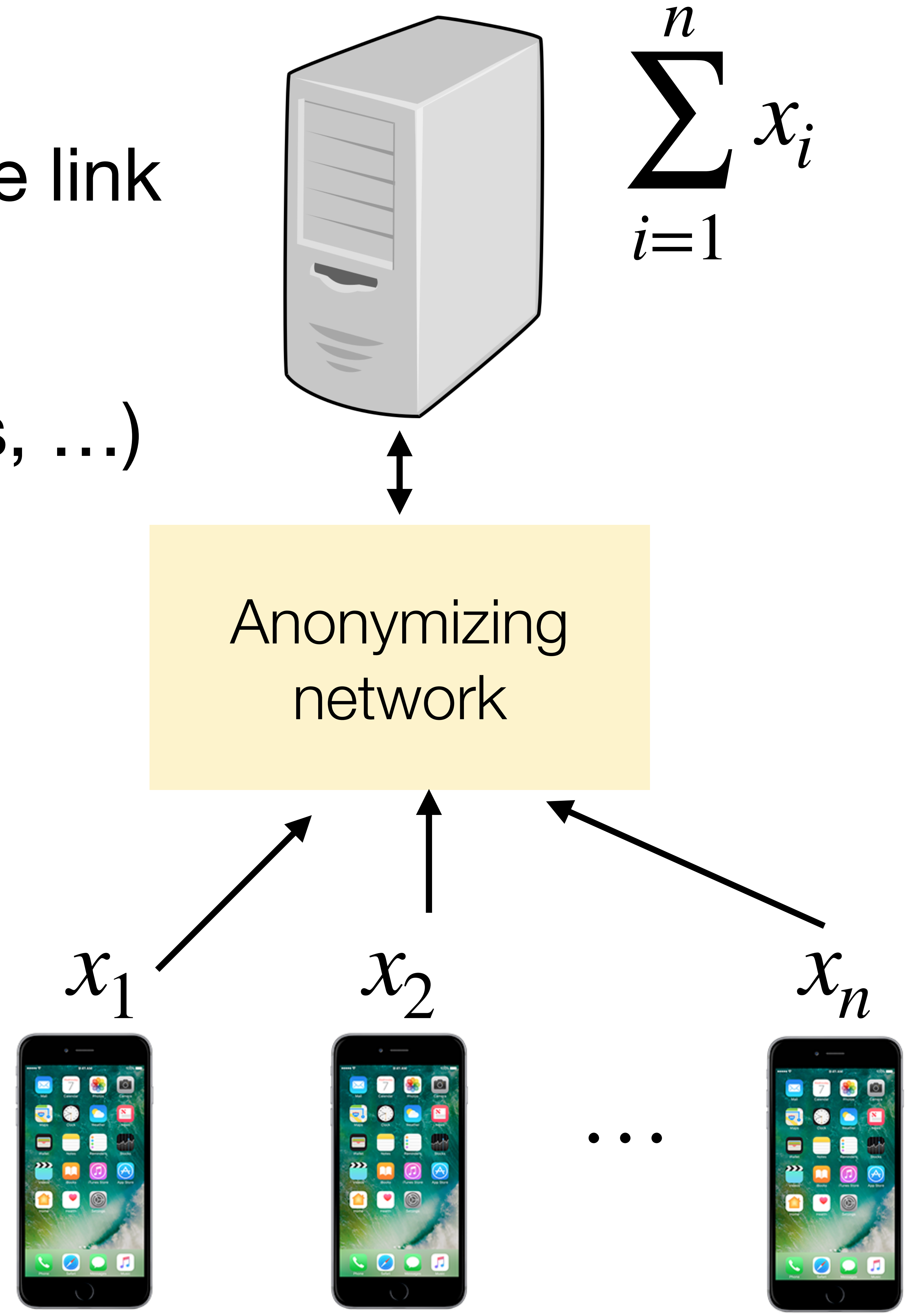
One approach

What tool from class could we use to hide the link between users and data submissions?

Anonymizing network (Tor, mix-nets, DC-nets, …)

Drawbacks?

- Systems either difficult to scale or vulnerable to traffic-analysis attacks



Another approach: randomized response

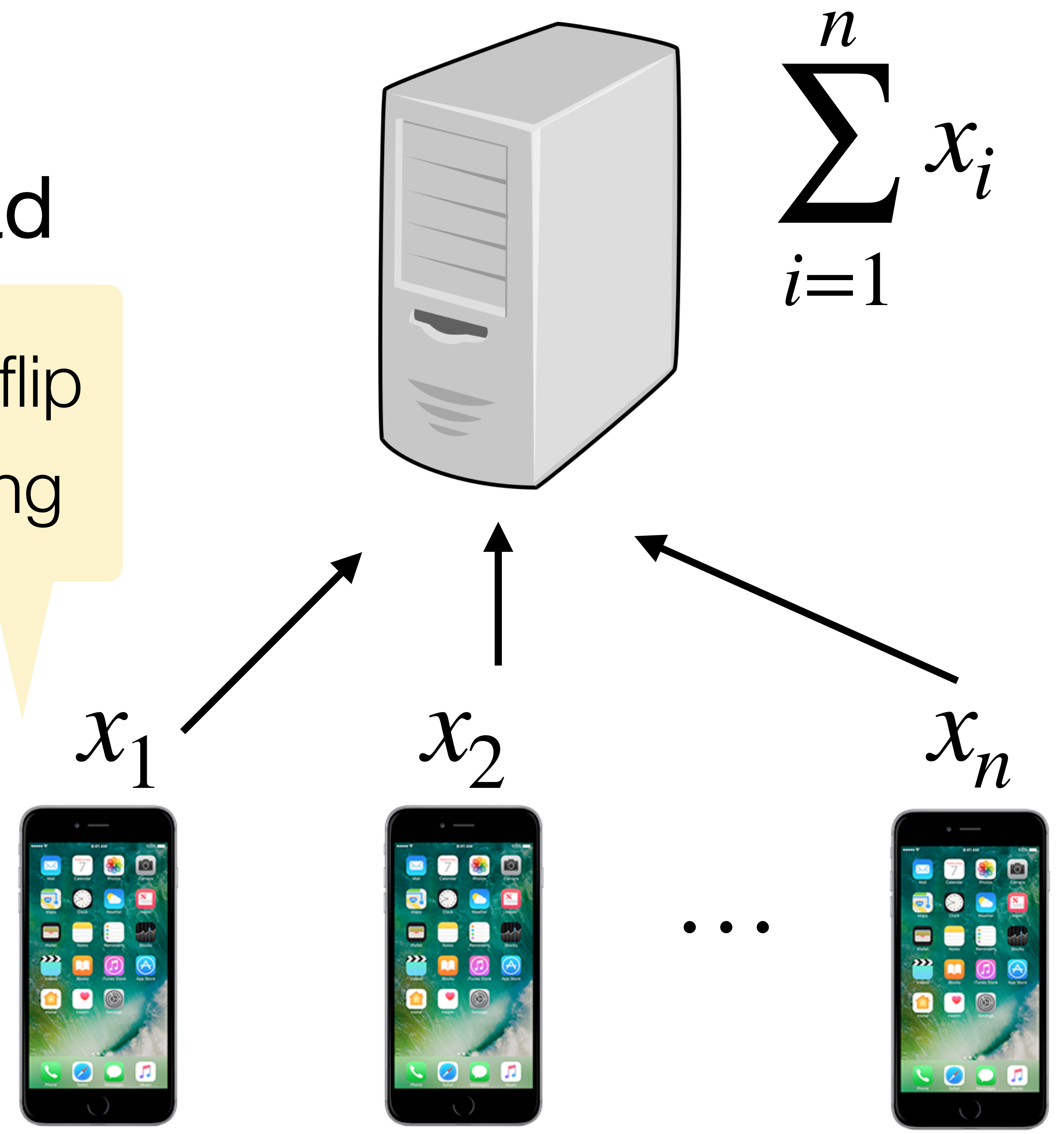
Advantages?

- Simple and scales well
- A malicious client can influence the final score by at most +/- 1: easy to detect a bad score

Disadvantages?

- Small p: weak privacy (can easily guess original input)
- Large p: not very useful

With probability p, flip bit x_i before sending



One more approach: additively homomorphic encryption

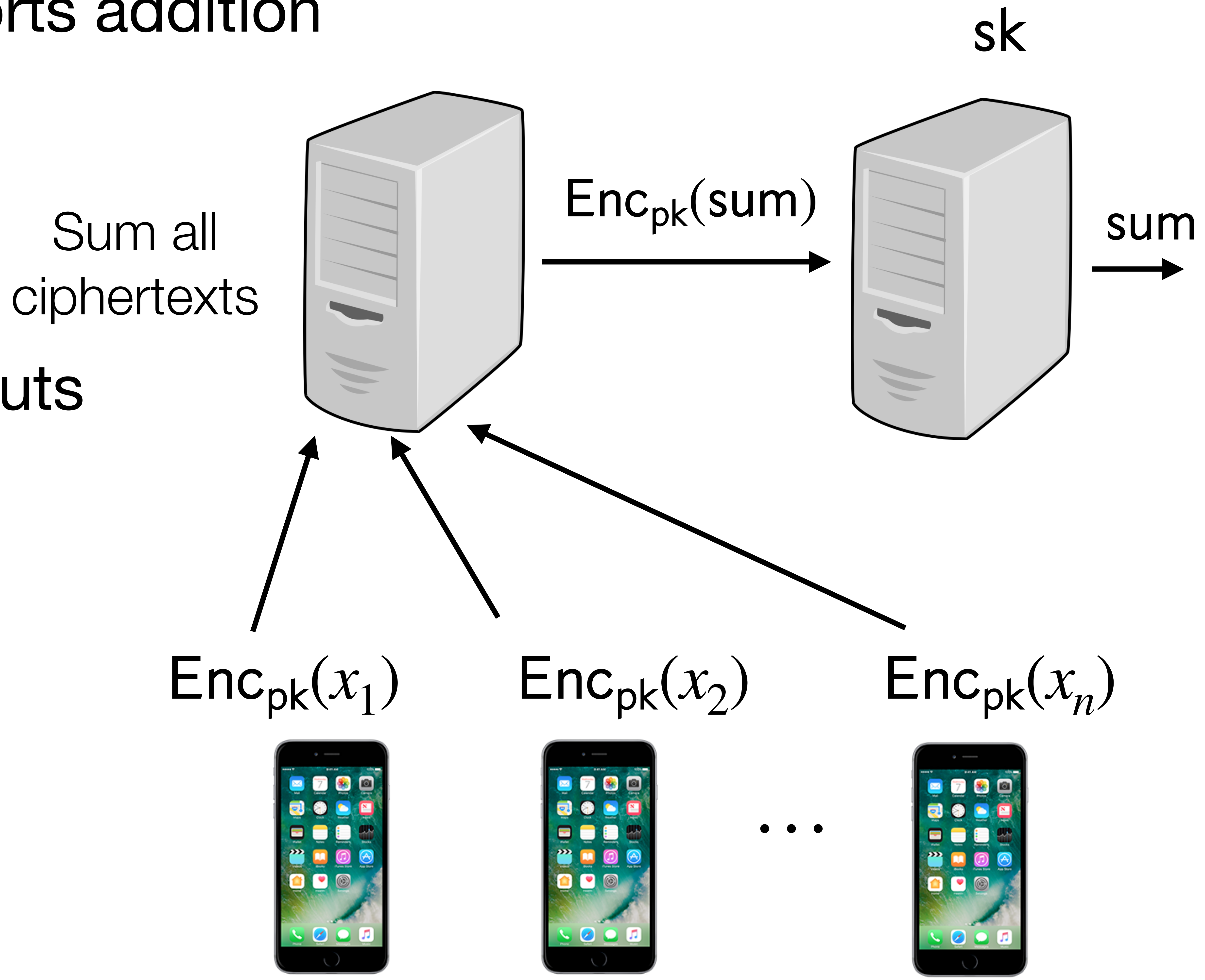
Tool: encryption scheme that supports addition

Advantages?

- Hides client contributions
- Correctly computes sum over inputs

Disadvantages?

- No protection against malicious client contributions
- No protection against malicious aggregator

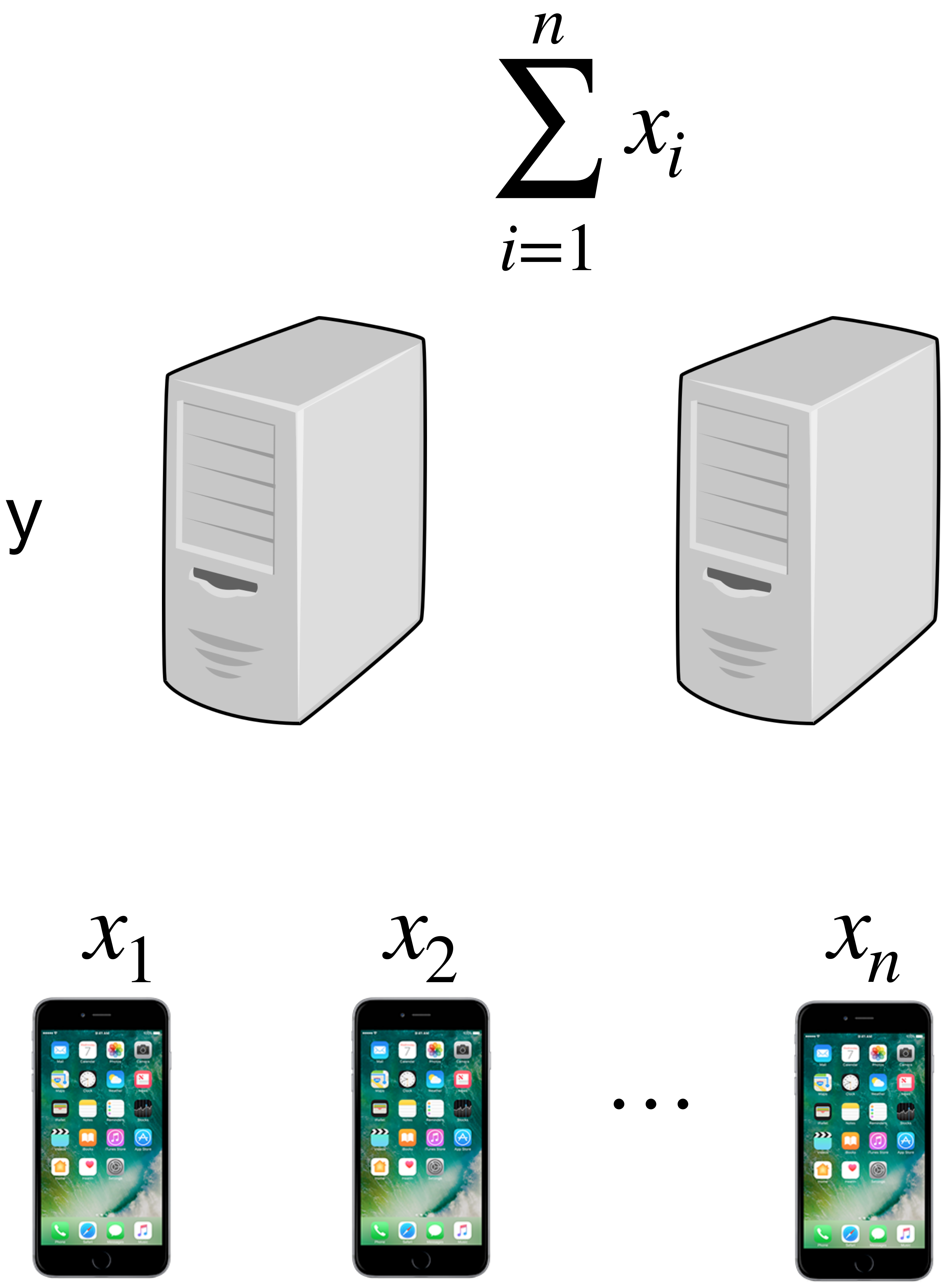


Prio

[Corrigan-Gibbs, Boneh]

All three of:

- Utility: Correctly computes sum
- Privacy: Hides user inputs
- Robustness: Malicious clients cannot overly influence sum



Prio properties

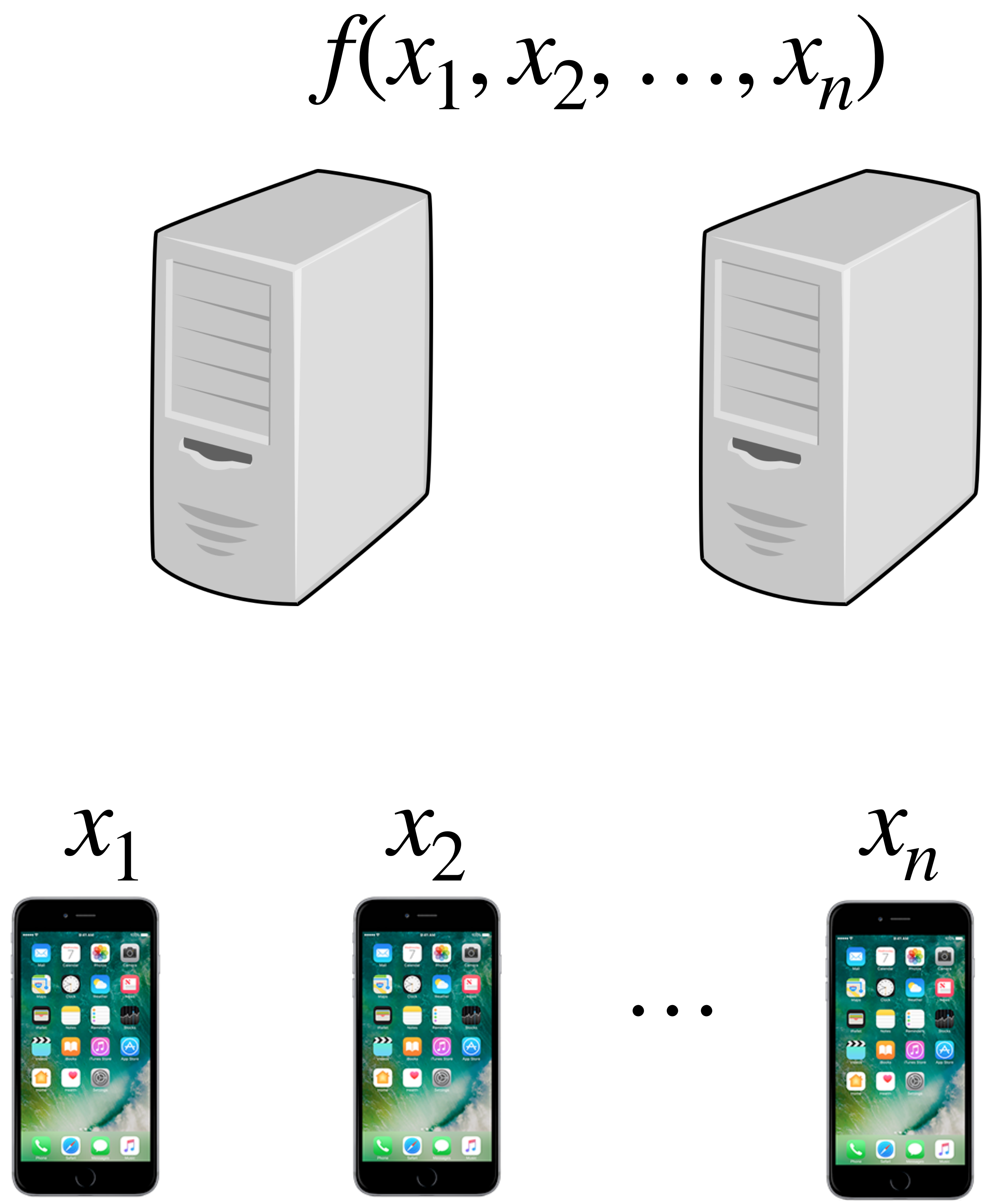
Aggregation function f

Correctness: If all servers are honest, servers learn $f(x_1, x_2, \ldots, x_n)$

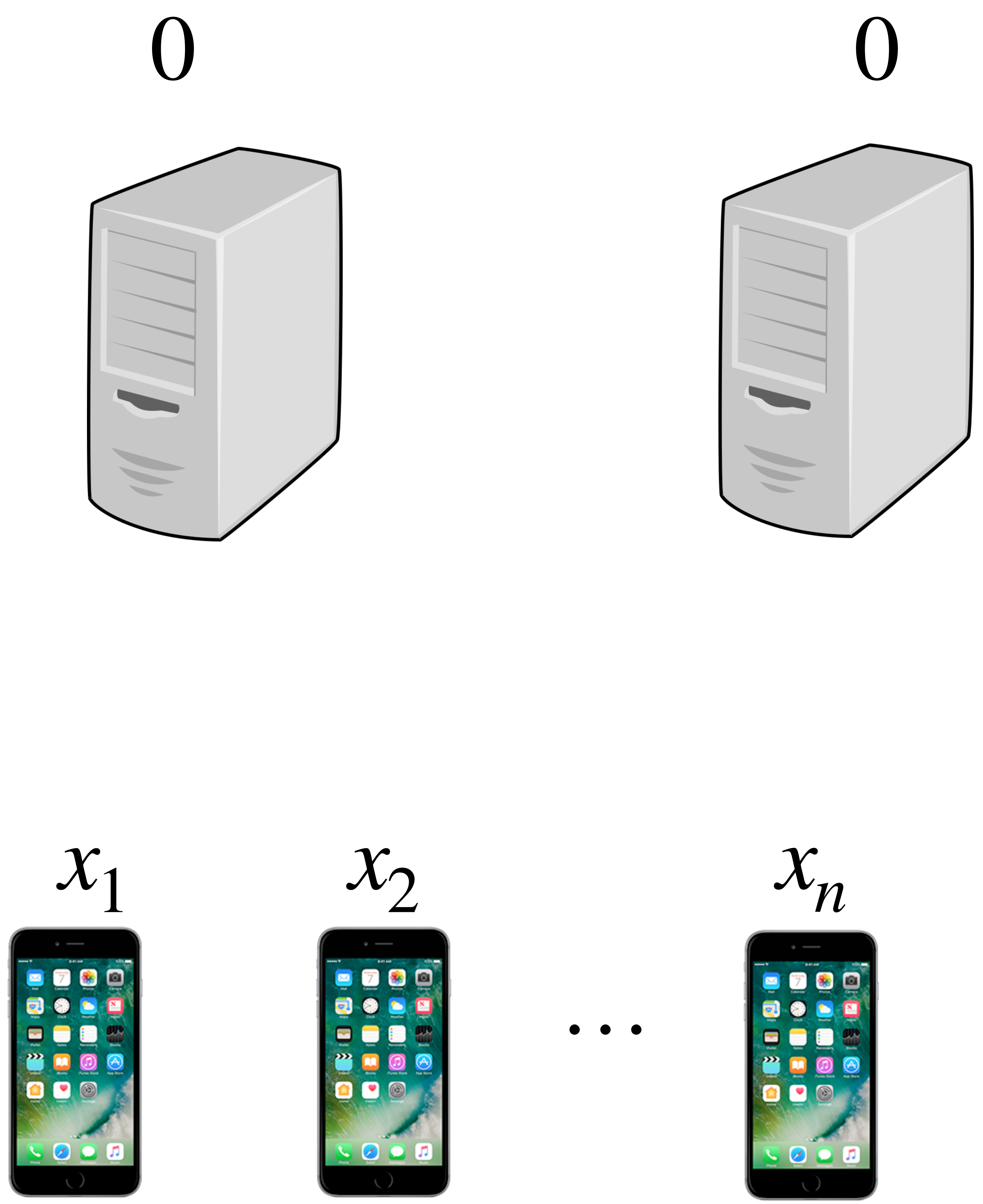
Privacy: If one server is honest, servers only learn $f(x_1, x_2, \ldots, x_n)$

- Privacy with 1 malicious server

Robustness: Malicious clients have bounded influence



Warm up: computing private sums

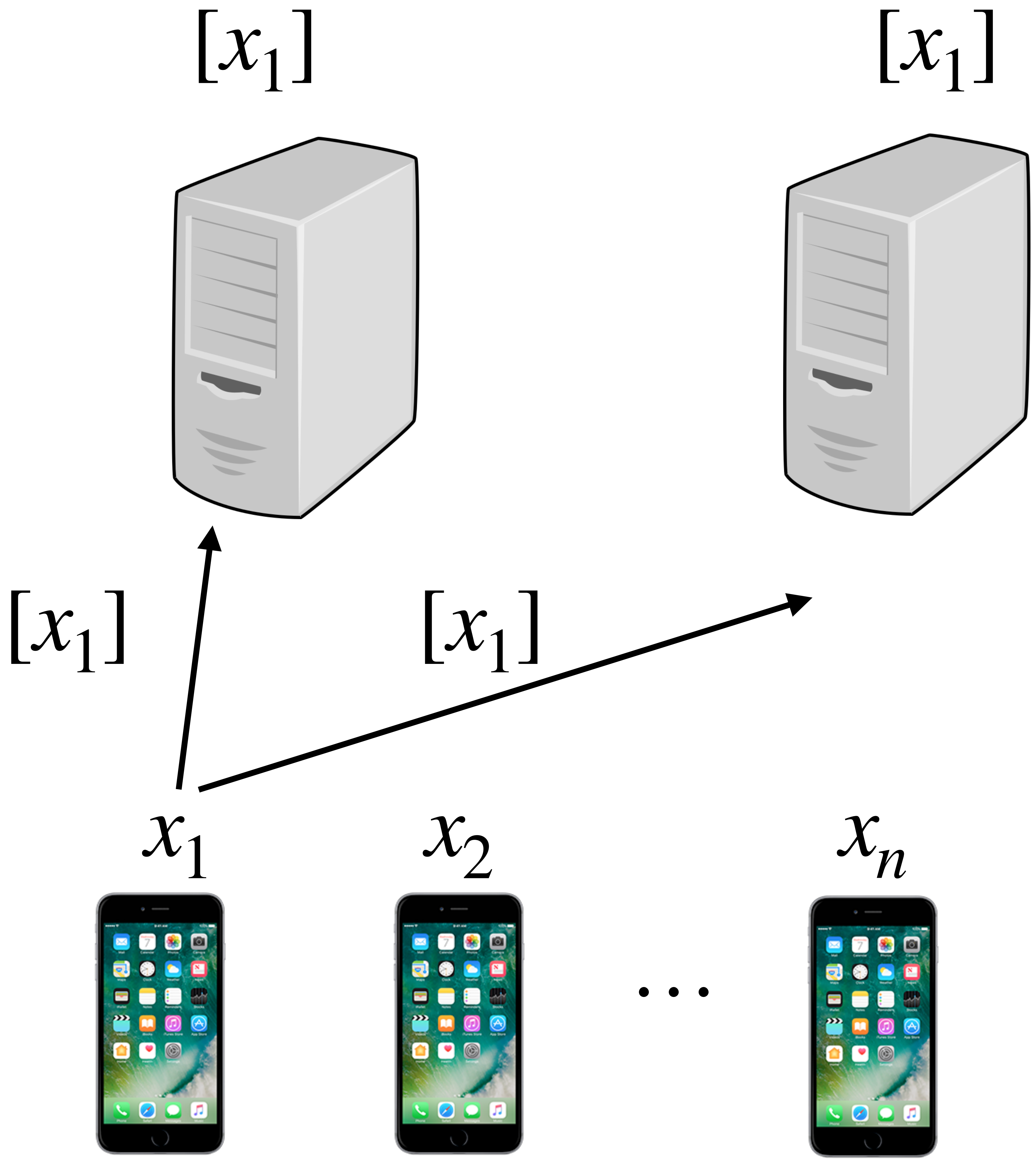


Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

Warm up: computing private sums

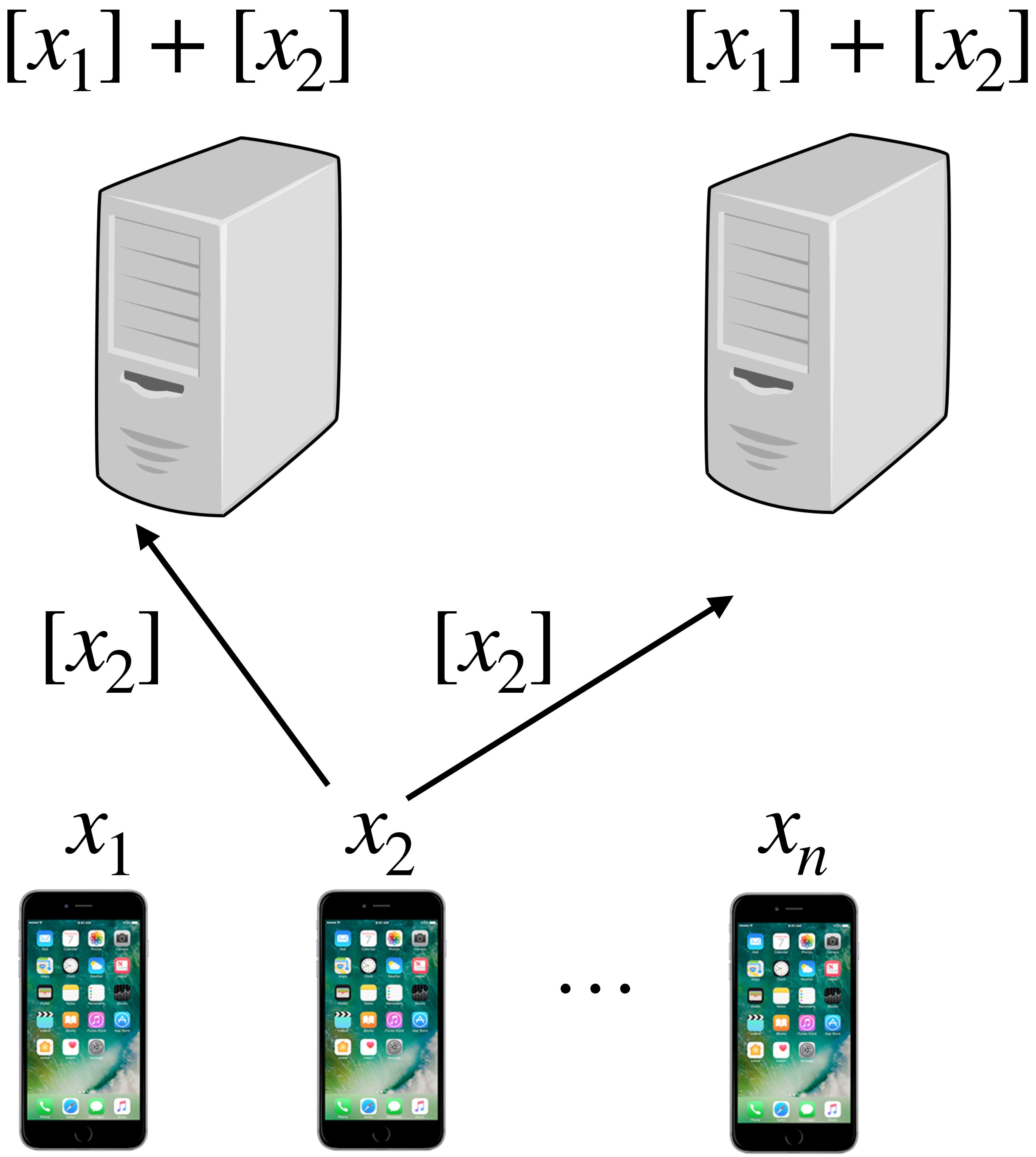


Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

Warm up: computing private sums

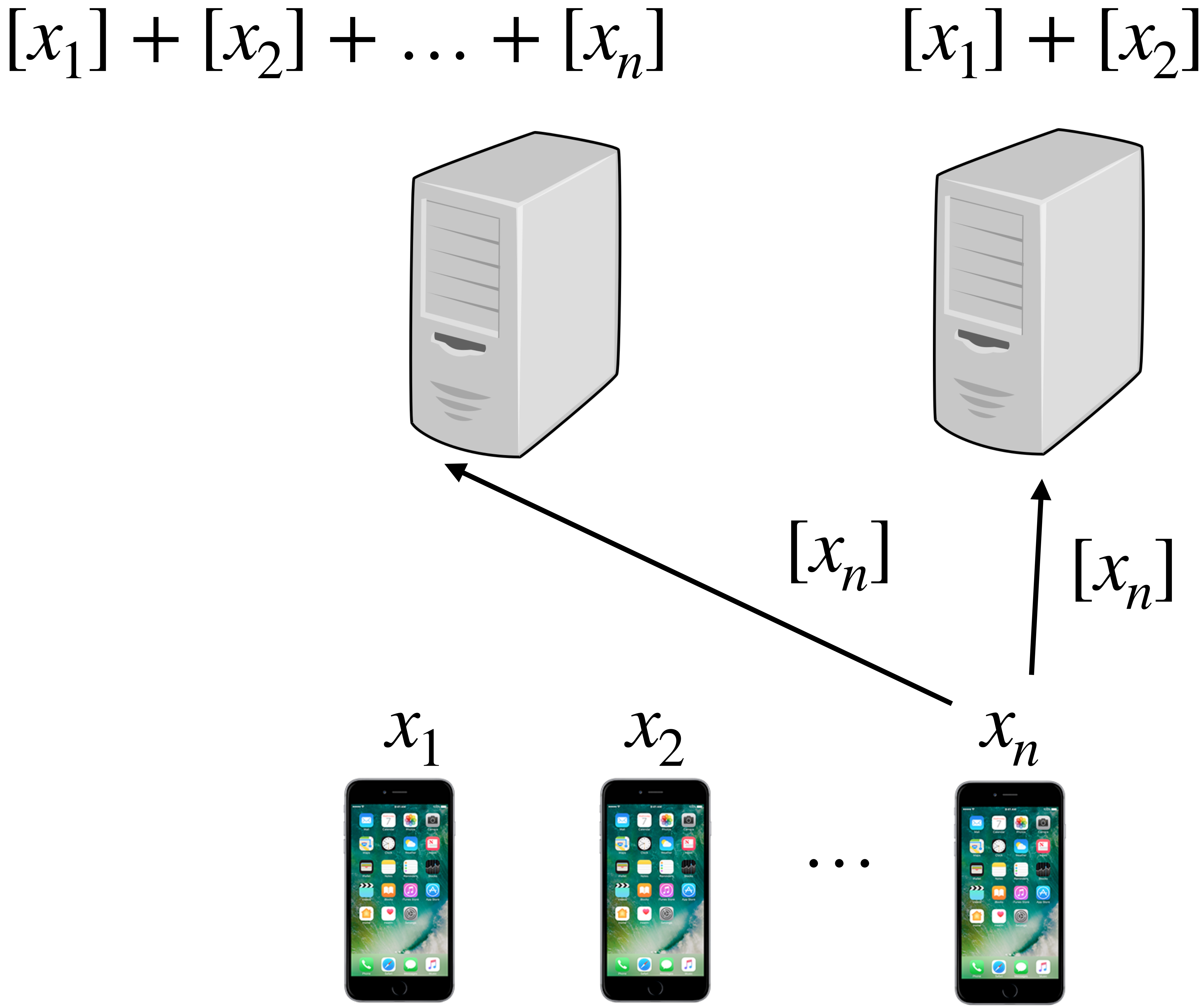


Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

Warm up: computing private sums

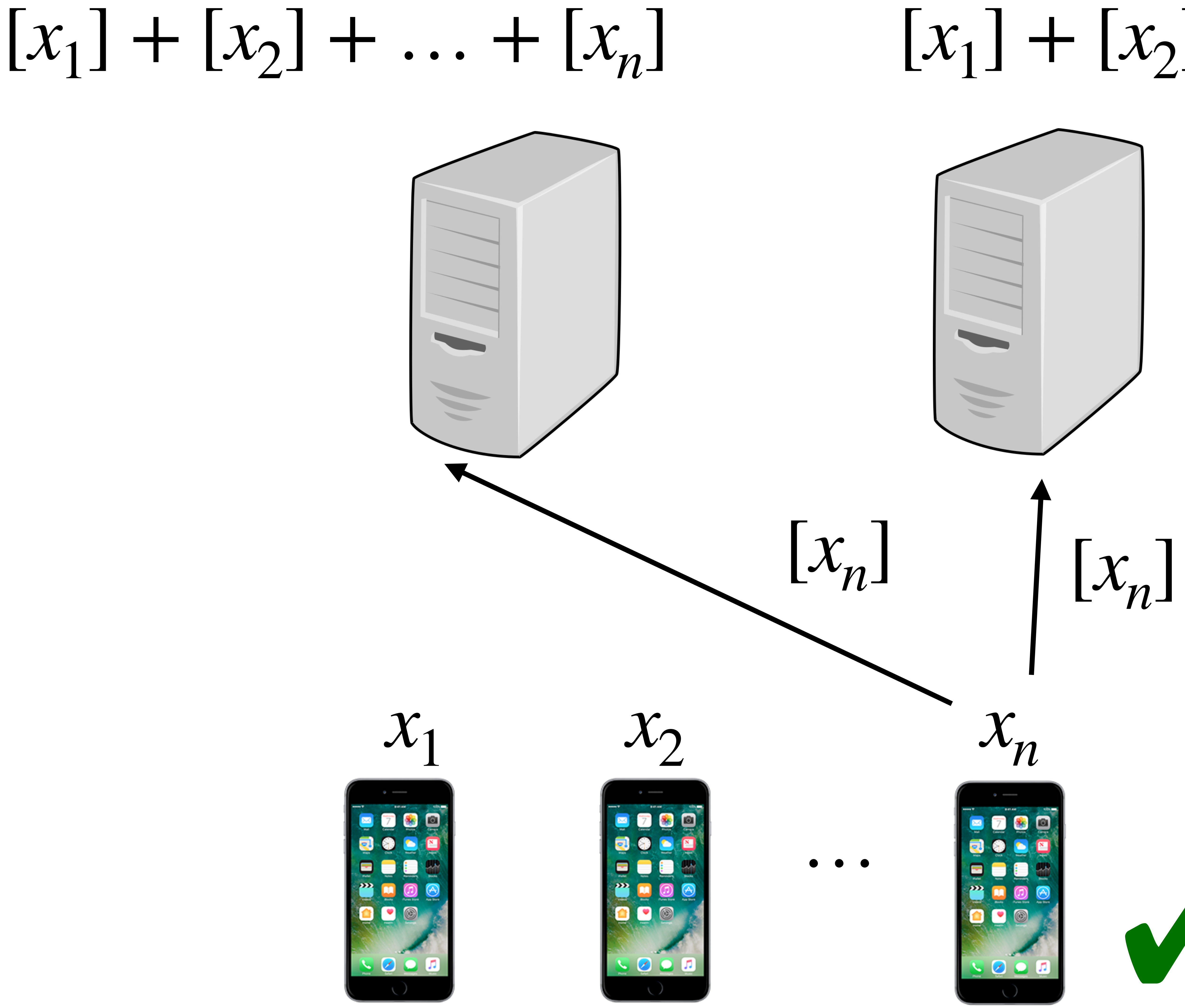


Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

Warm up: computing private sums



Mozilla wants to know:

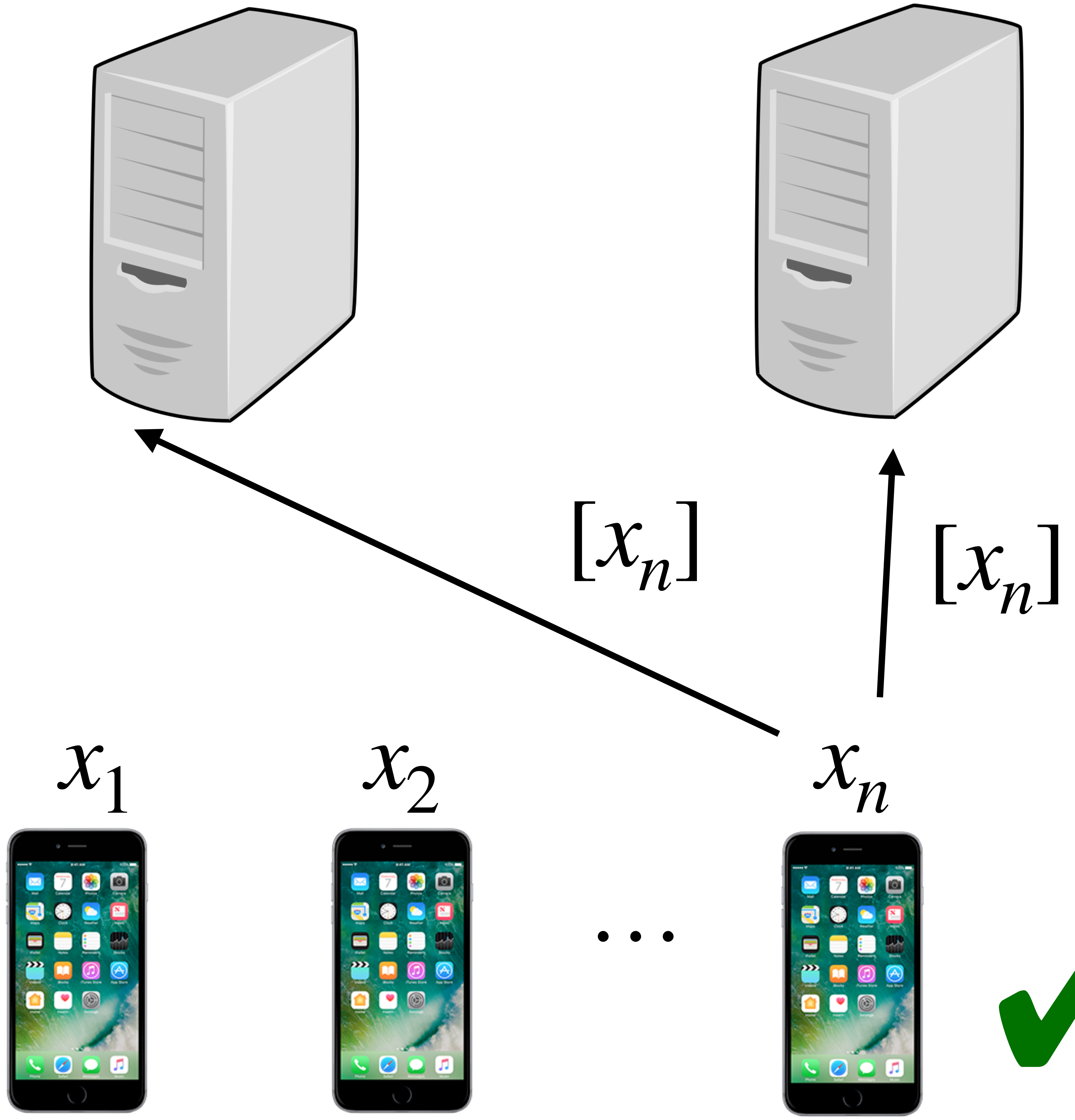
- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

✓ **Correctness:** If all servers are honest, servers learn $f(x_1, x_2, \ldots, x_n)$

Warm up: computing private sums

$[x_1] + [x_2] + \ldots + [x_n]$ $\quad$ $[x_1] + [x_2] + \ldots + [x_n]$



Mozilla wants to know:

- How many users disable content blocking on `nytimes.com`?

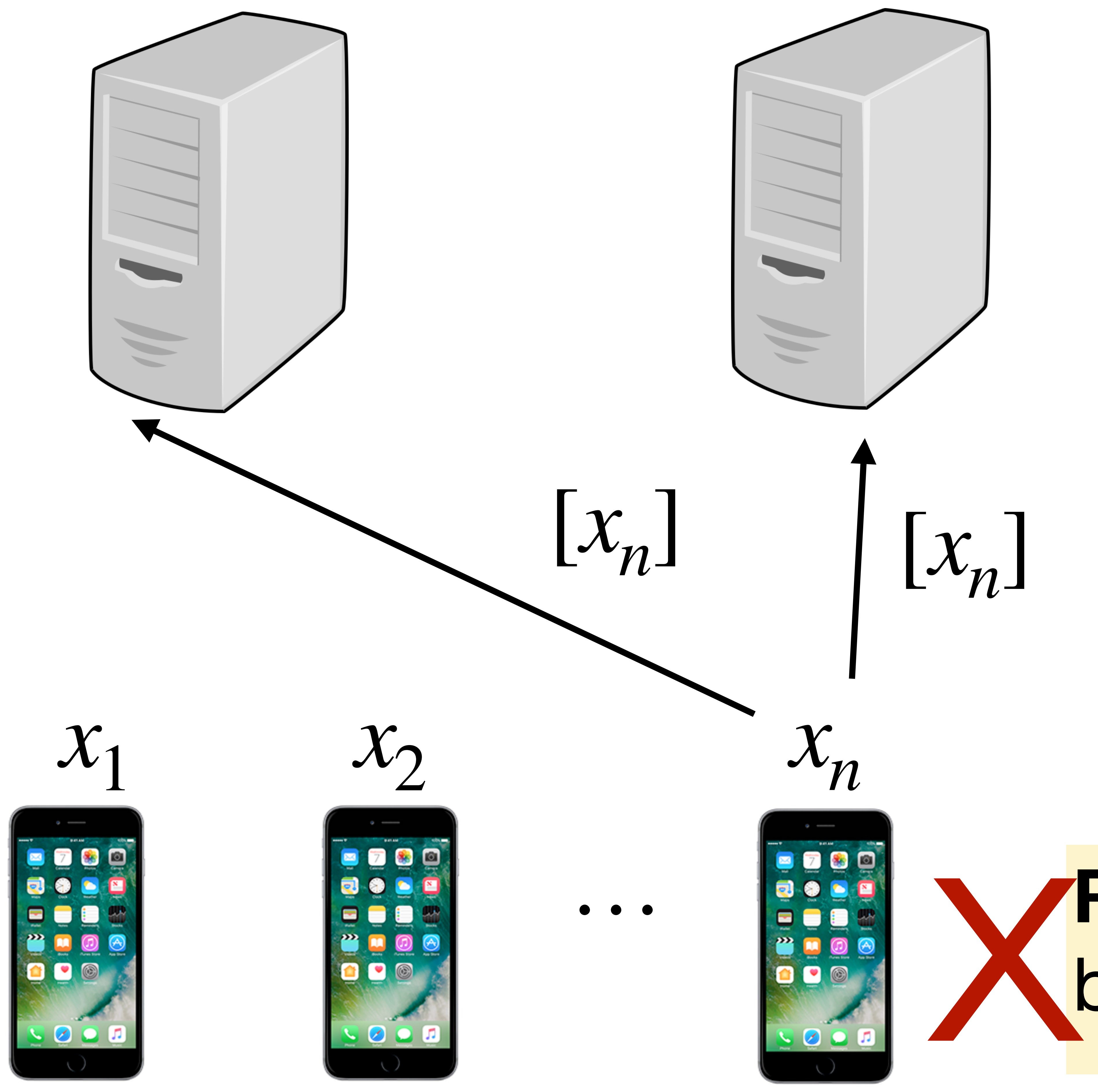
Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

✓ **Privacy:** If one server is honest, servers only learn $f(x_1, x_2, \ldots, x_n)$

- Privacy with 1 malicious server

Warm up: computing private sums

$[x_1] + [x_2] + \ldots + [x_n]$ $\qquad$ $[x_1] + [x_2] + \ldots + [x_n]$



Mozilla wants to know:

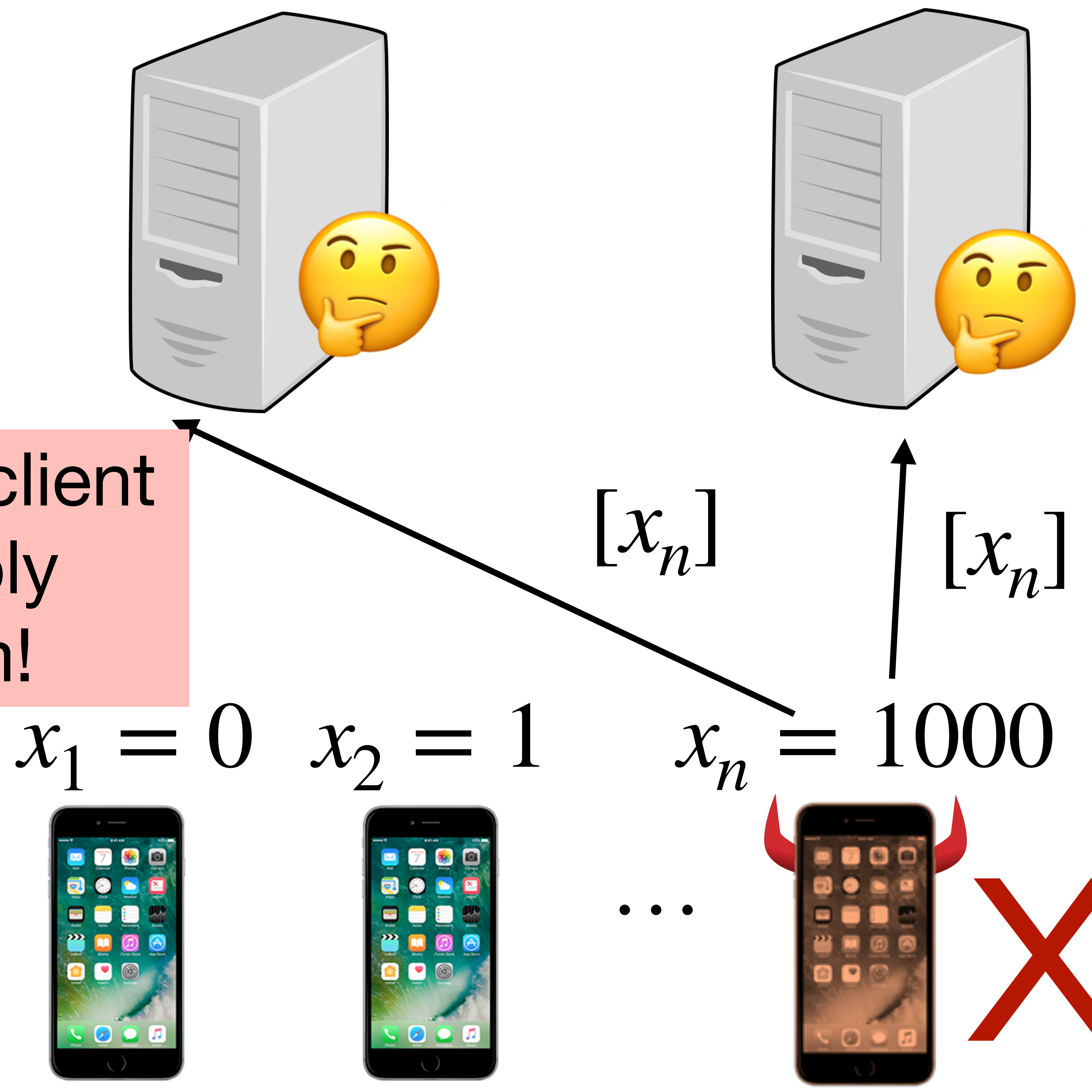
- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

Robustness: Malicious clients have bounded influence

Warm up: computing private sums

$[x_1] + [x_2] + \ldots + [x_n]$ $[x_1] + [x_2] + \ldots + [x_n]$



Mozilla wants to know:

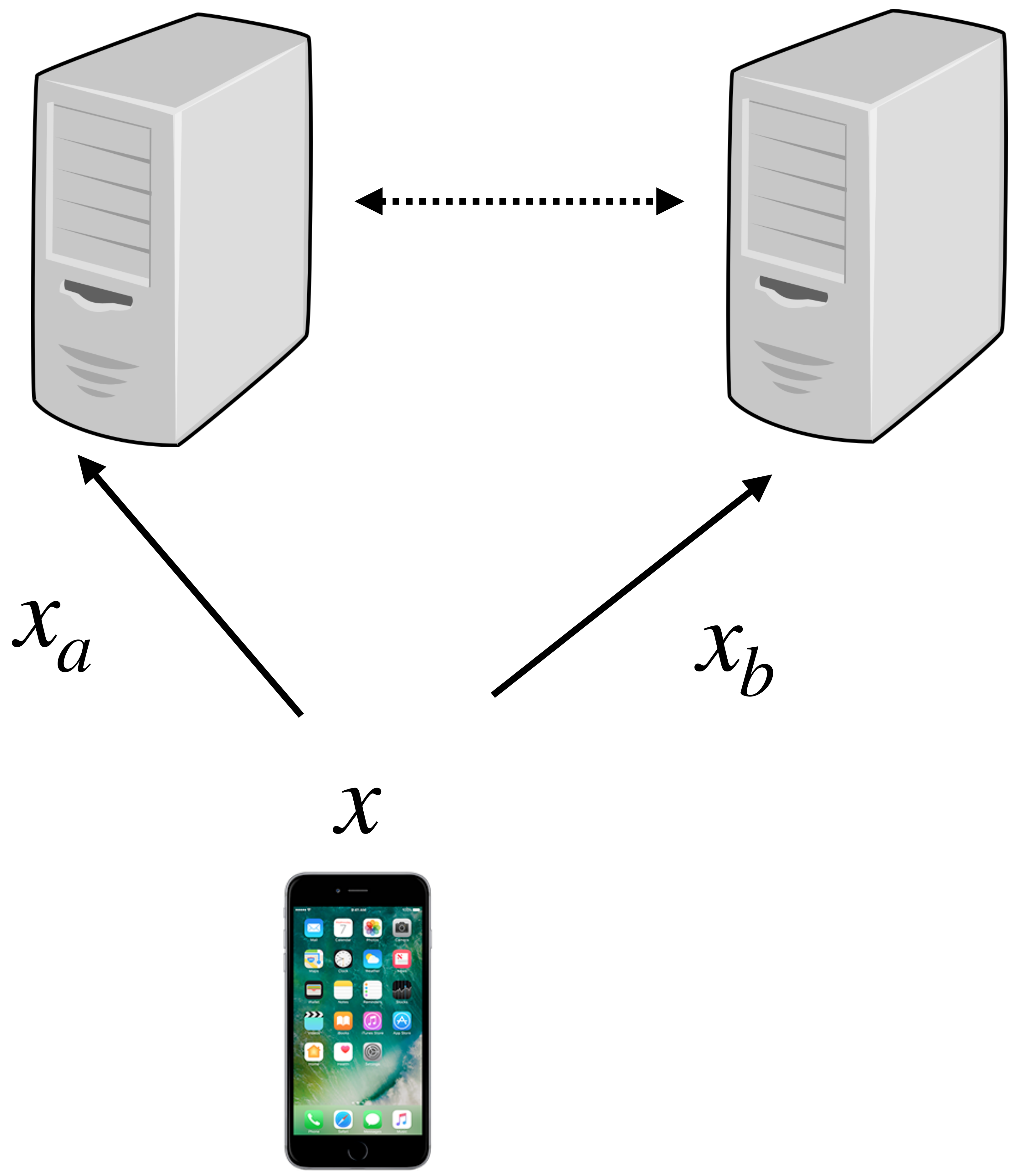
- How many users disable content blocking on `nytimes.com`?

Each client i has x_i where $x_i = 1$ if content blocking is disabled on nytimes.com and $x_i = 0$ otherwise

One malicious client can undetectably corrupt the sum!

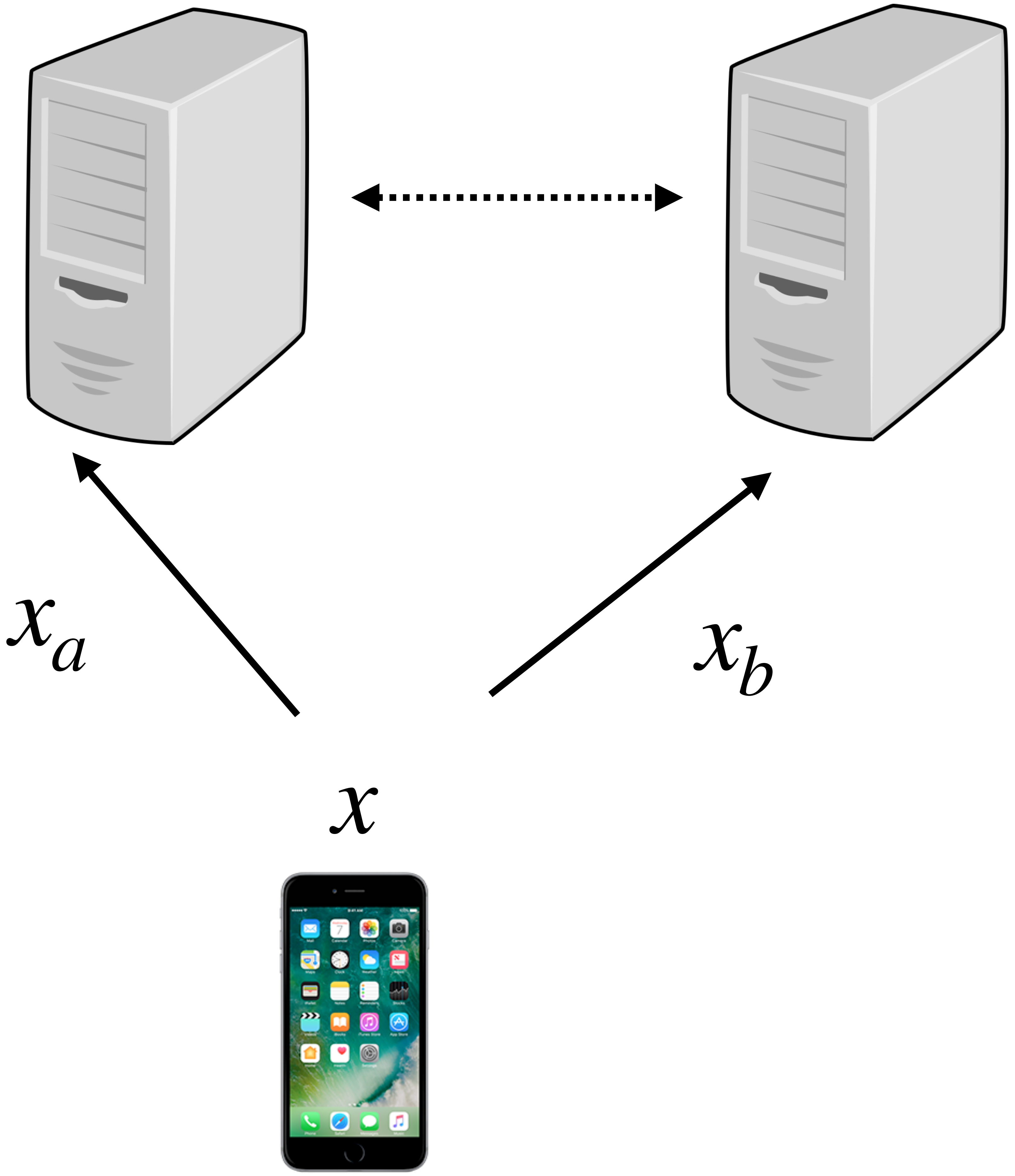
Robustness: Malicious clients have bounded influence

Secret-shared non-interactive proofs (SNIPs)



Goal: Servers want to learn if $x_a + x_b \in \{0,1\}$ without learning x

Secret-shared non-interactive proofs (SNIPs)



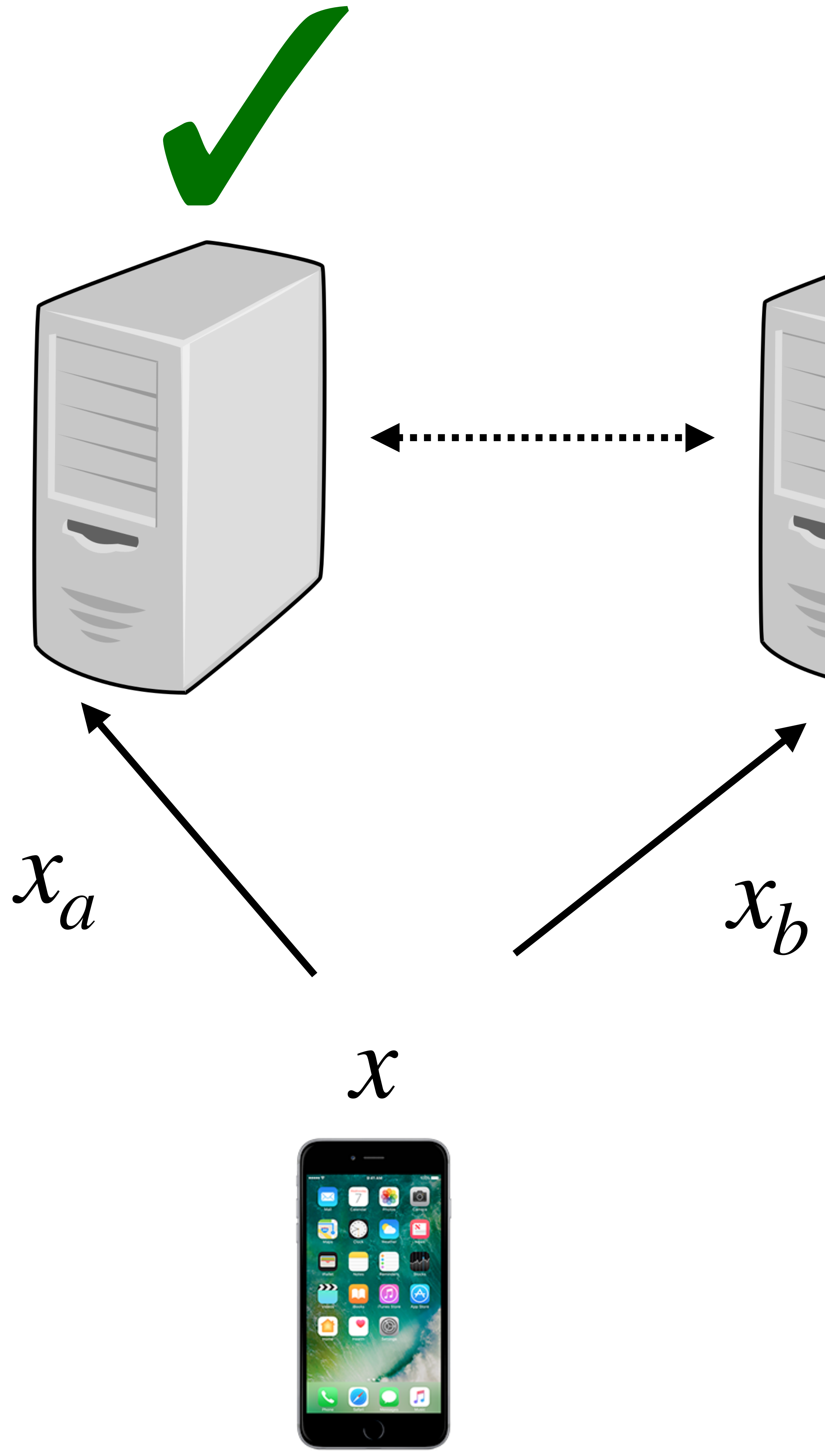
More generally, servers

- Hold shares of the client's private value x
- Hold an arbitrary public predicate $\mathsf{Valid}(\cdot)$
- Want to test if $\mathsf{Valid}(x) = 1$

EX: $\mathsf{Valid}(x) =$ "$x \in \{0,1\}$"

What other Valid predicates might be useful?

Secret-shared non-interactive proofs (SNIPs)

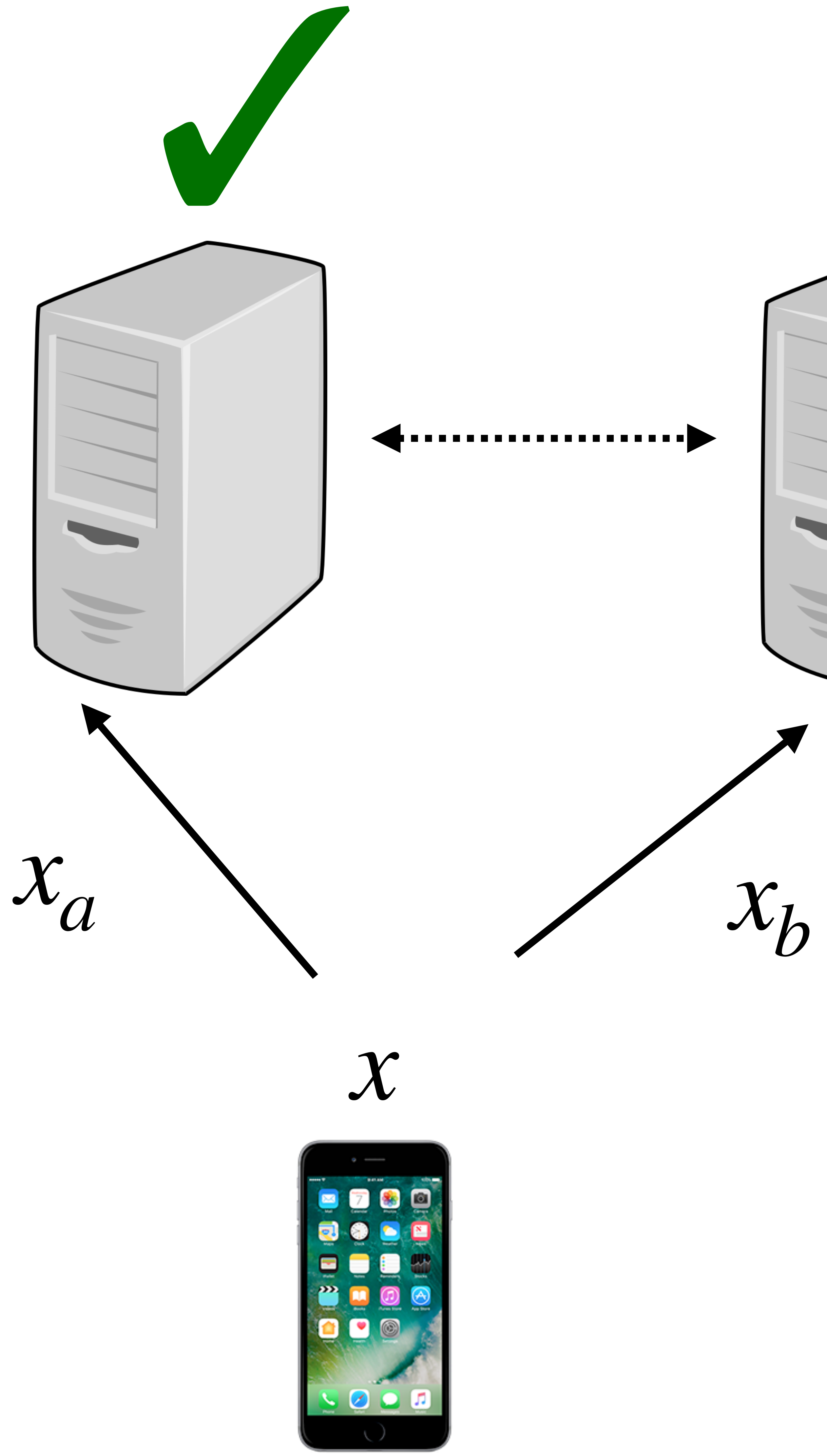


More generally, servers

- Hold shares of the client's private value x
- Hold an arbitrary public predicate $\mathsf{Valid}(\cdot)$
- Want to test if $\mathsf{Valid}(x) = 1$

EX: $\mathsf{Valid}(x) =$ "$x \in \{0,1\}$"

Secret-shared non-interactive proofs (SNIPs)



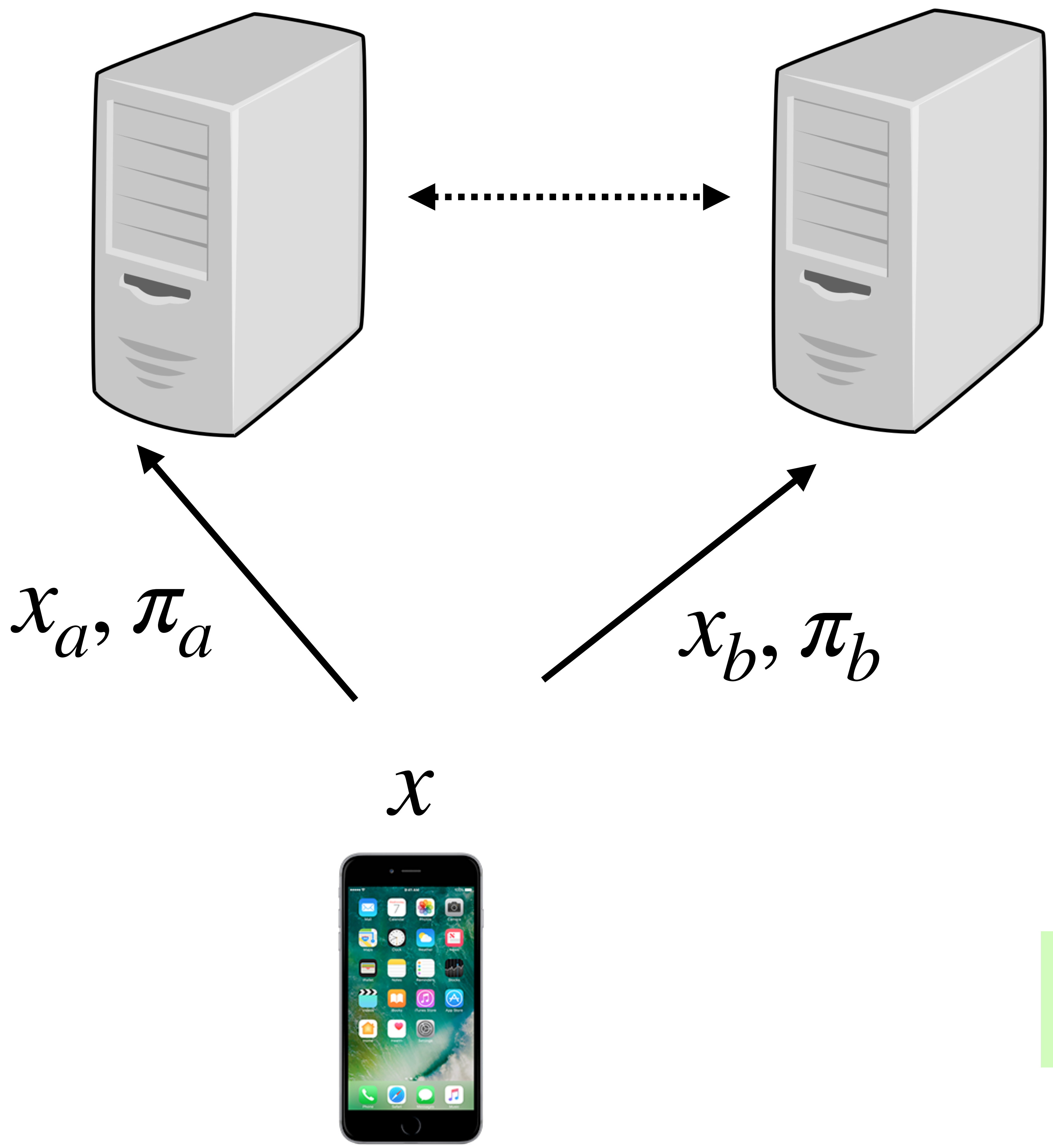
More generally, servers

- Hold shares of the client's private value x
- Hold an arbitrary public predicate $\mathsf{Valid}(\,\cdot\,)$
- Want to test if $\mathsf{Valid}(x) = 1$

EX: $\mathsf{Valid}(x) =$ "$x \in \{0,1\}$"

Prio servers detect and reject malformed client input.
Each client can influence final sum by at most +/- 1.

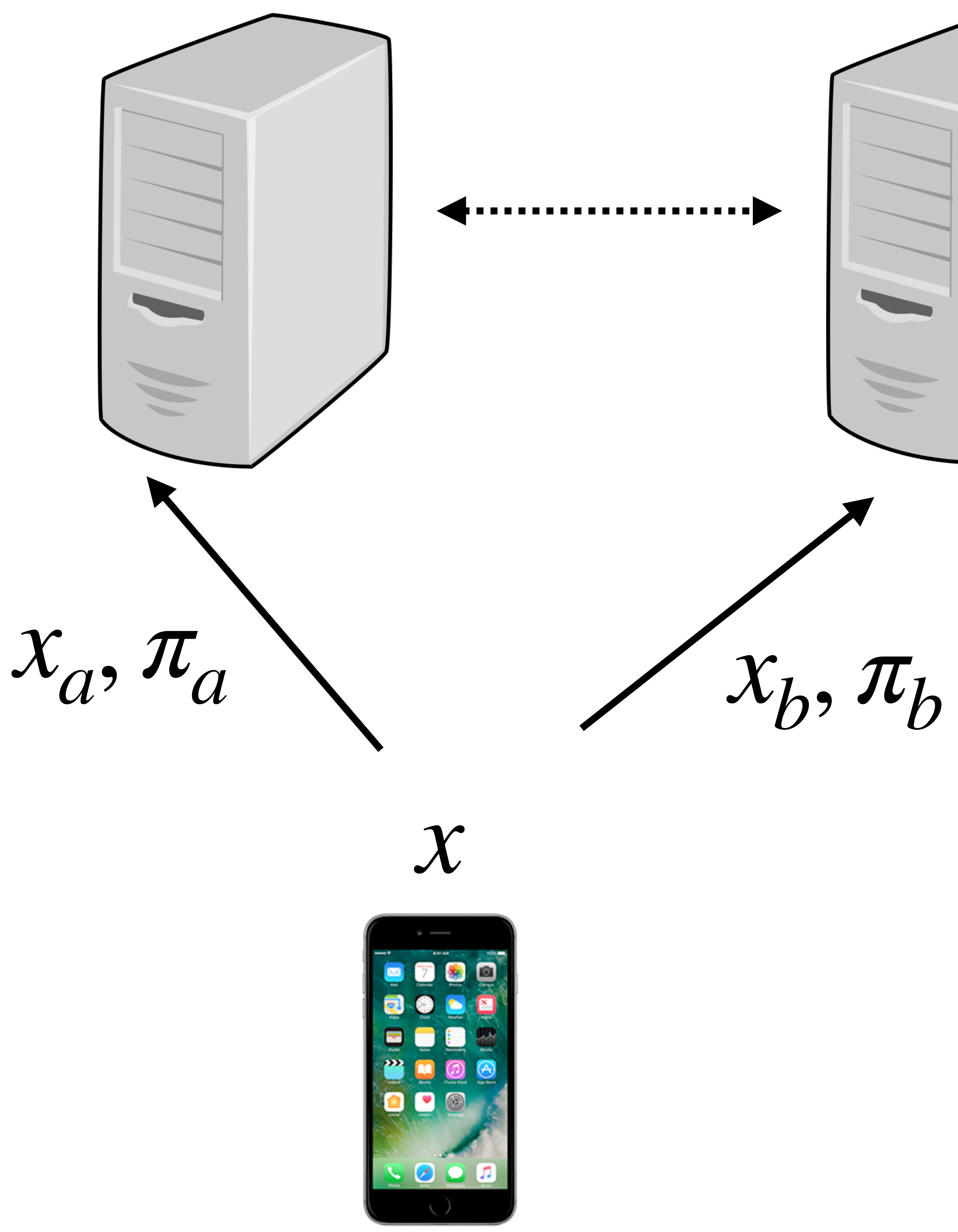
How do SNIPs work?



Servers could run MPC to check that
Valid$(x) = 1$

Idea: Client can make servers' job easier!

How do SNIPs work?

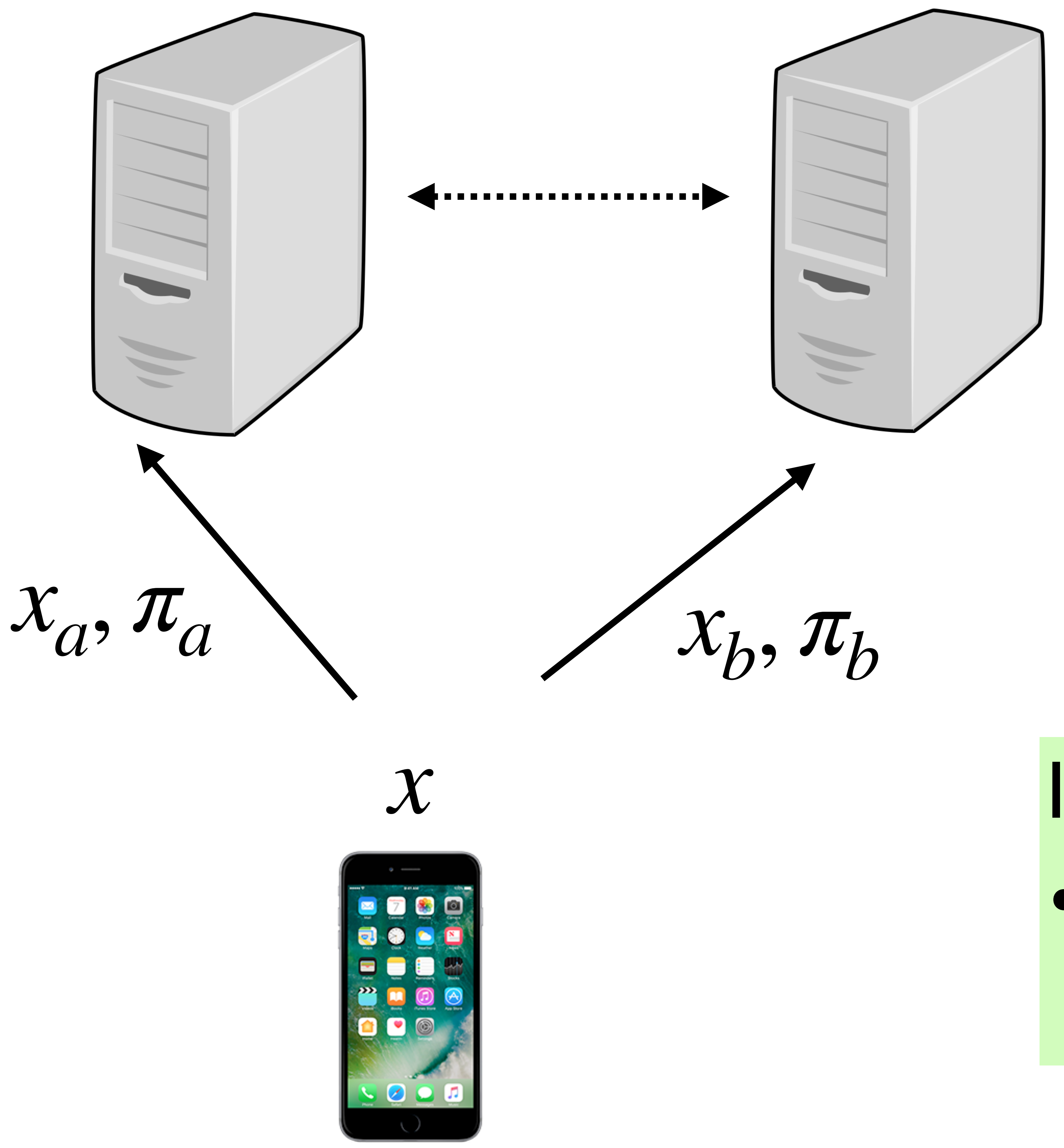


Servers could run MPC to check that
$\mathsf{Valid}(x) = 1$

Idea: Client can make servers' job easier!

- Client generates the transcripts that servers *would* have observed in MPC

How do SNIPs work?



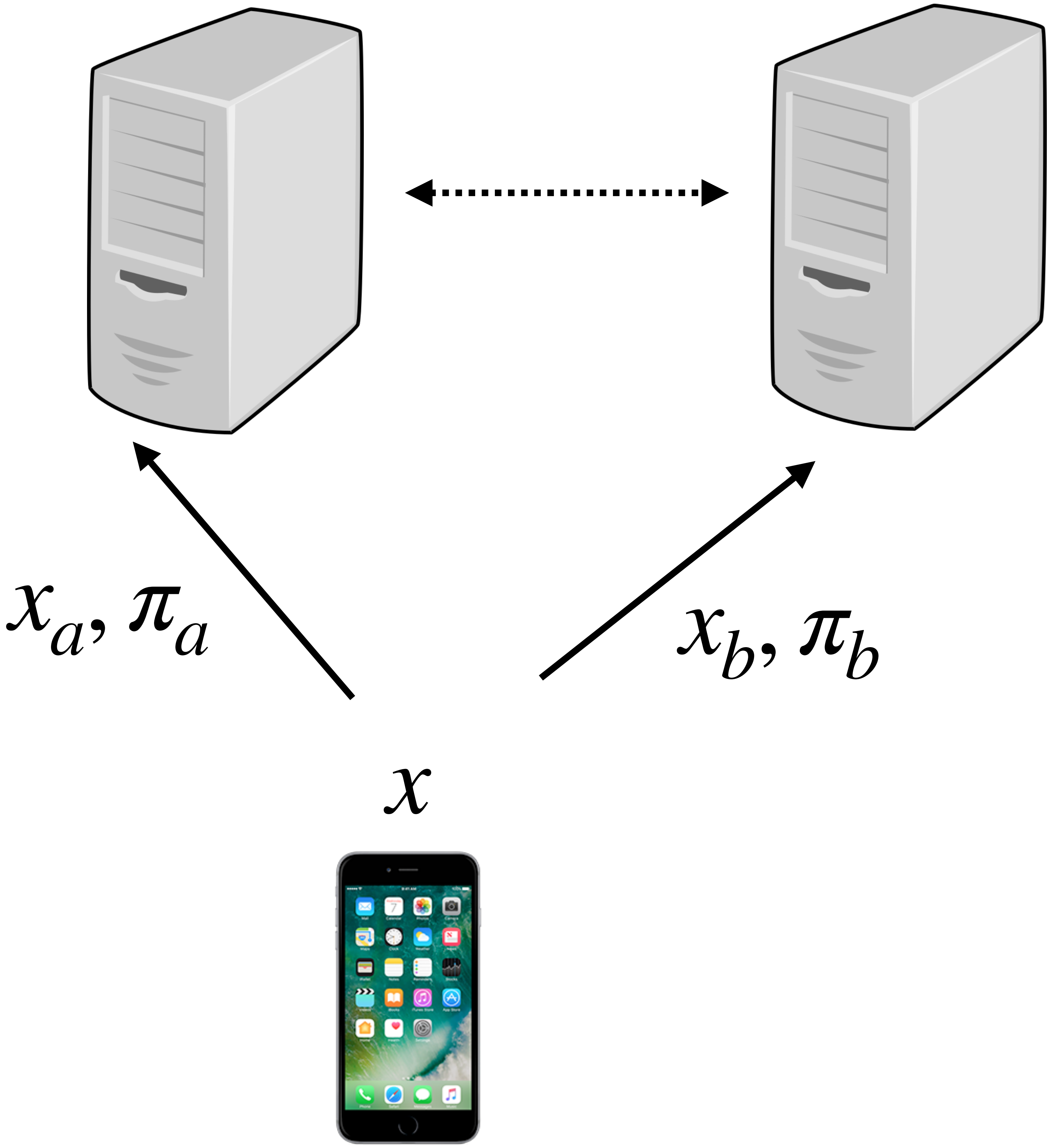
Servers check that the transcripts are valid and consistent.

- Checking a transcript is much easier than generating it

Idea: Client can make servers’ job easier!

- Client generates the transcripts that servers *would* have observed in MPC

How do SNIPs work?



Recall in MPC:

- Addition is free
- Multiplication requires communication

Goal: Use client to efficiently check multiplications

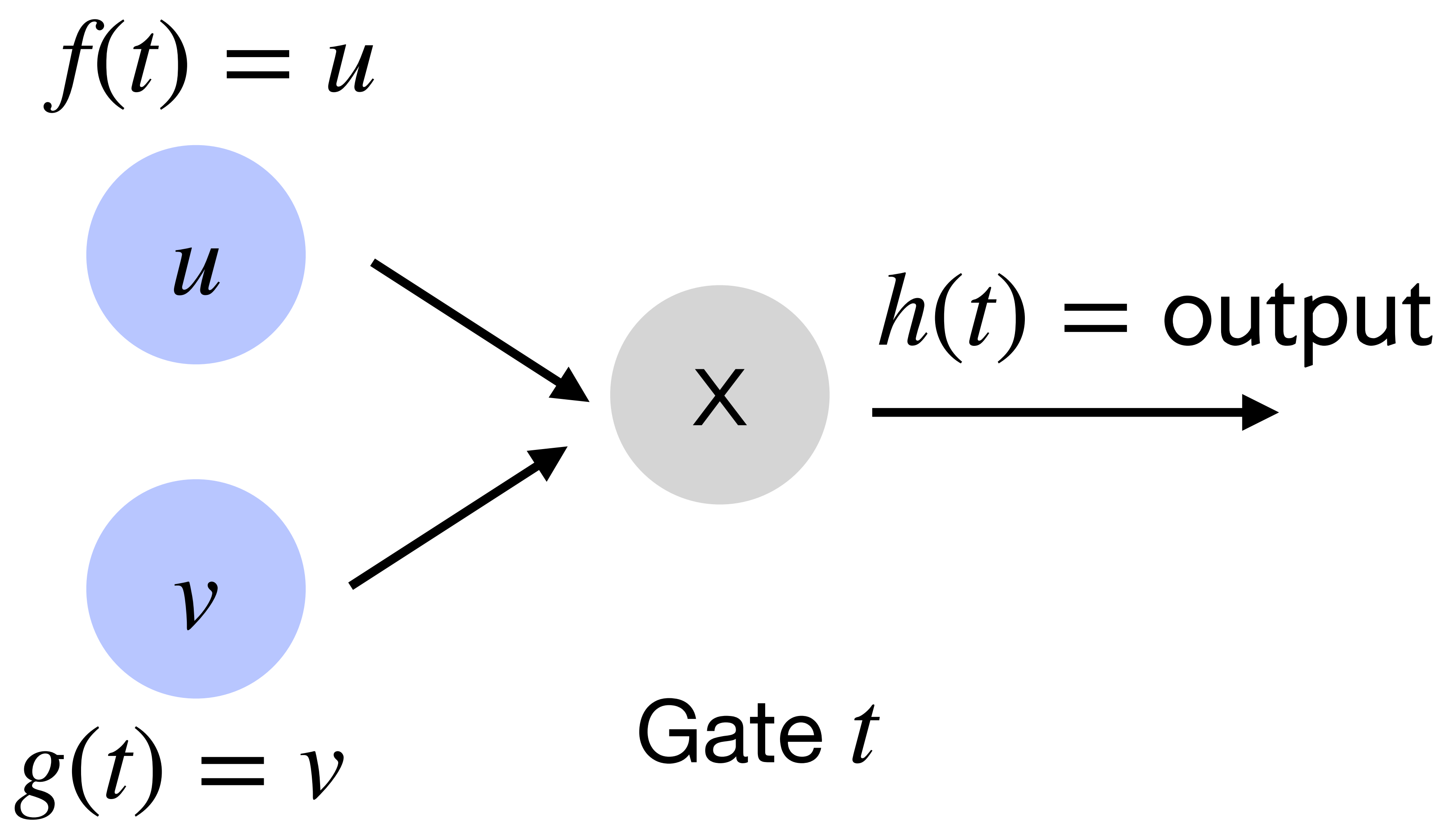
How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$ Degree $m - 1$
   - $g(t) = v_t$ Degree $m - 1$
3. Let $h = f \cdot g$ Degree $2m - 2$
4. Secret-share coefficients of h to 2 servers

Goal: prove $\text{Valid}(x) = 1$

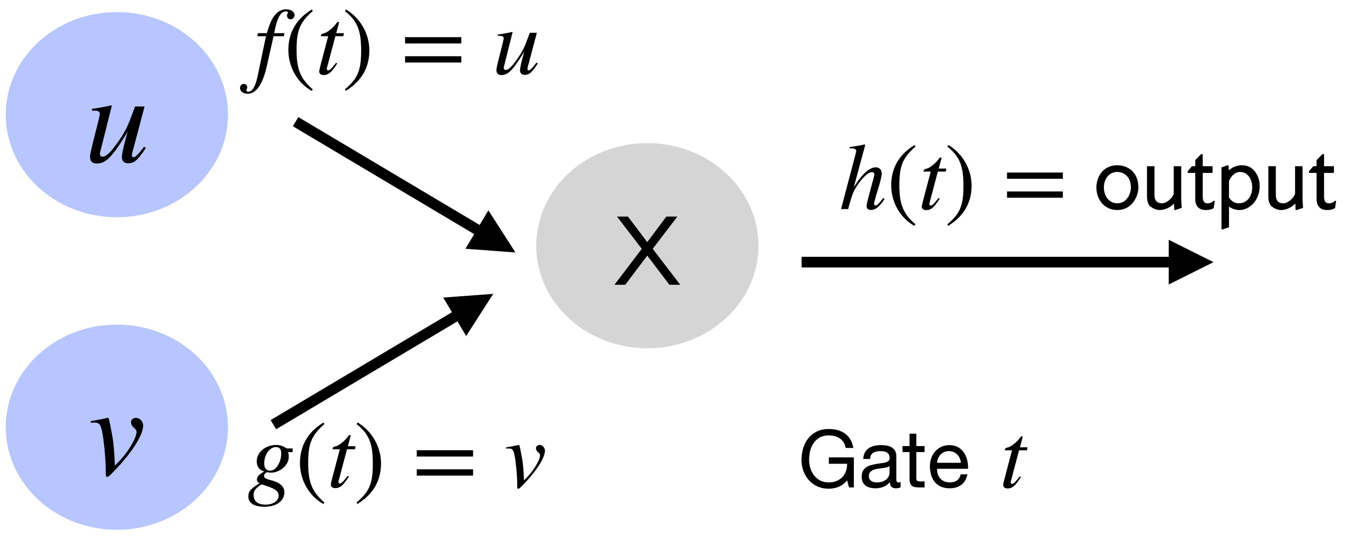
Number of multiplication gates: m



How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

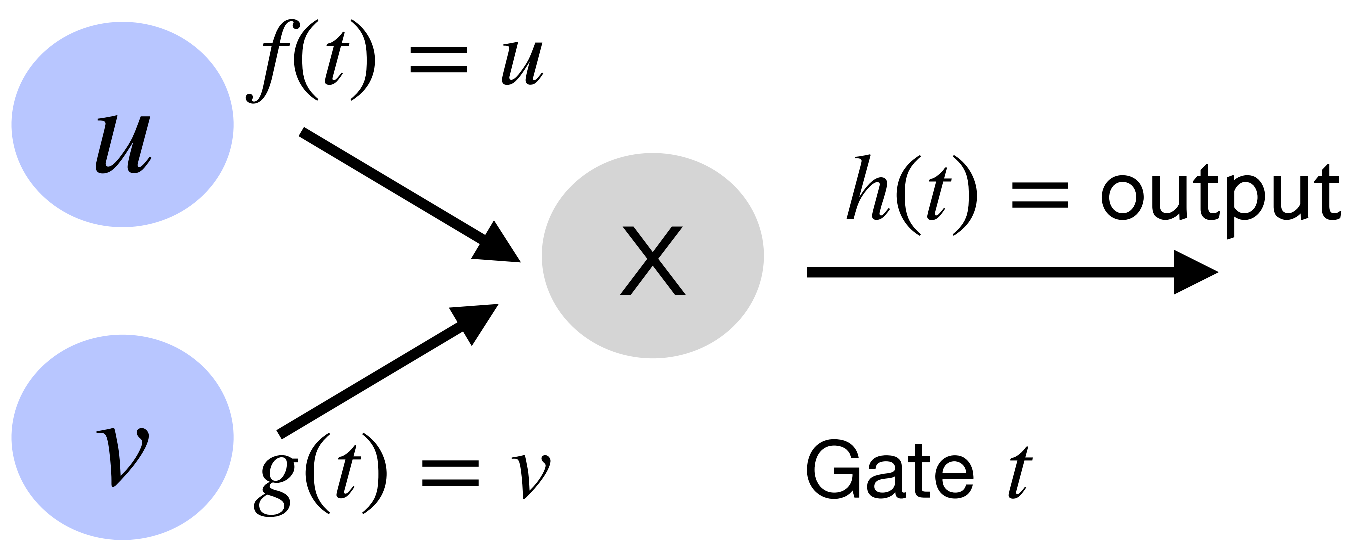
Server verification

1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

Server verification

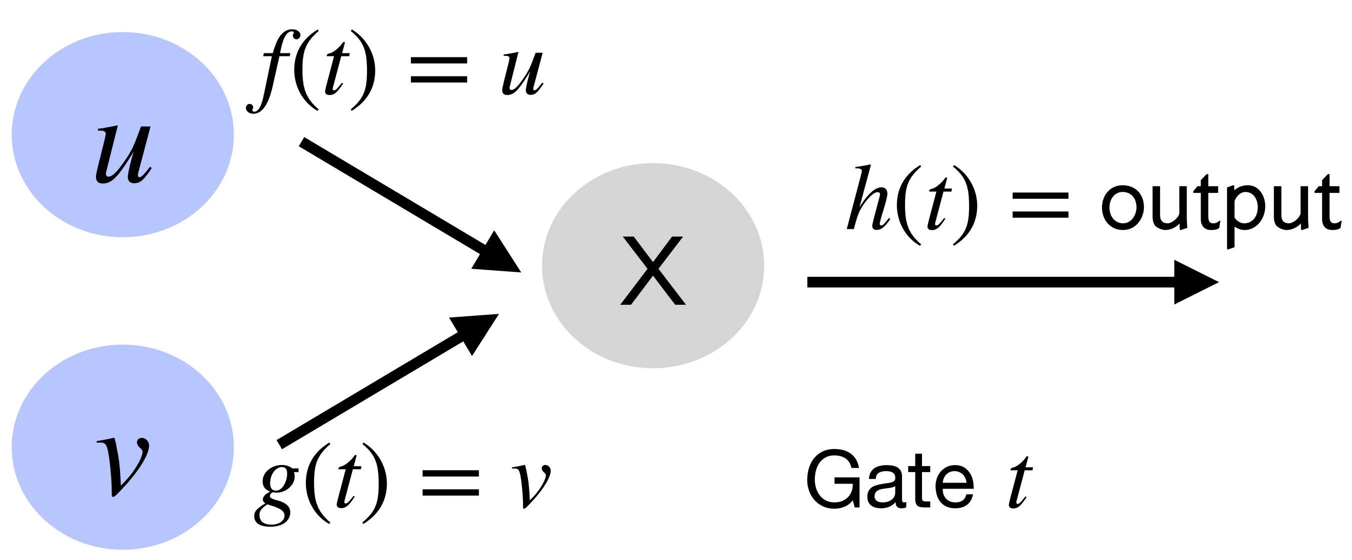
1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g

Question: How can the servers tell if the client sent the wrong h?

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

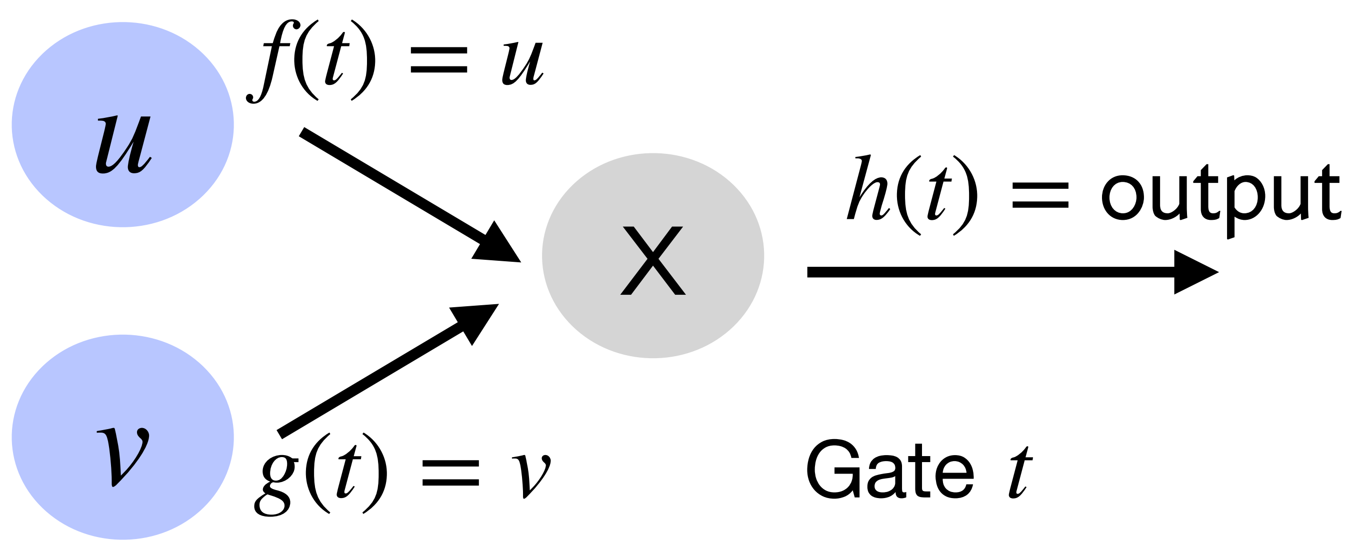
Server verification

1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g
3. Check that h is well-formed $(f \cdot g = h)$
   - Servers choose random r
   - Evaluate $\alpha \leftarrow f(r) \cdot g(r) - h(r)$
   - If $f \cdot g \neq h$, $\Pr[\alpha = 0]$ is negligible
   - Checking this requires 1 multiplication of secret-shared values

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

Probability that client cheated and didn't get caught is negligible

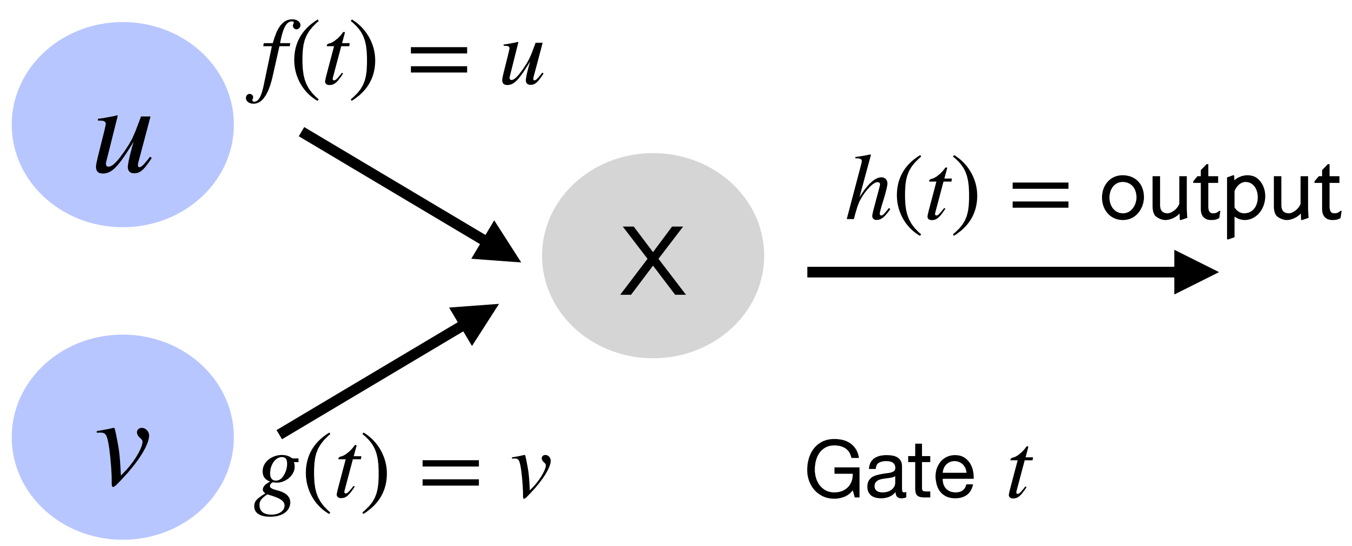
Server verification

1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g
3. Check that h is well-formed $(f \cdot g = h)$
   - Servers choose random r
   - Evaluate $\alpha \leftarrow f(r) \cdot g(r) - h(r)$
   - If $f \cdot g \neq h$, $\Pr[\alpha = 0]$ is negligible
   - Checking this requires 1 multiplication of secret-shared values

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

Due to the fact that polynomial of at most degree $2m - 2$ can have at most $2m - 2$ zeroes (m is number of multiplication gates)

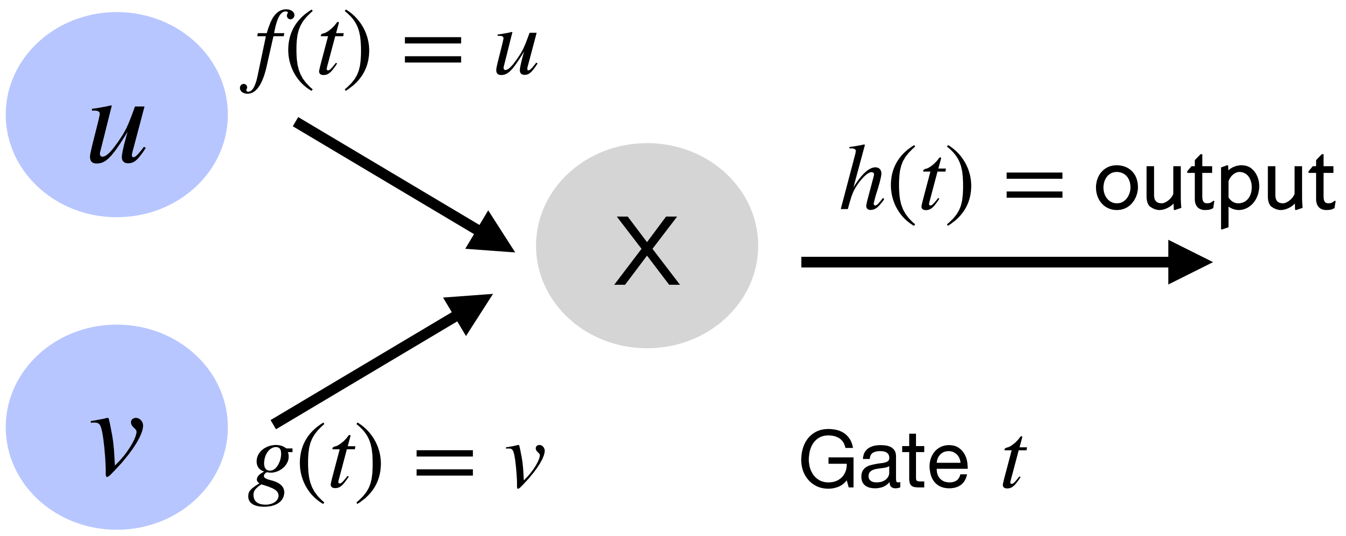
Server verification

1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g
3. Check that h is well-formed $(f \cdot g = h)$
   - Servers choose random r
   - Evaluate $\alpha \leftarrow f(r) \cdot g(r) - h(r)$
   - If $f \cdot g \neq h$, $\Pr[\alpha = 0]$ is negligible
   - Checking this requires 1 multiplication of secret-shared values

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$

Client can provide Beaver triple to servers.

- If client sends malformed Beaver triple, servers still catch cheating client with overwhelming probability

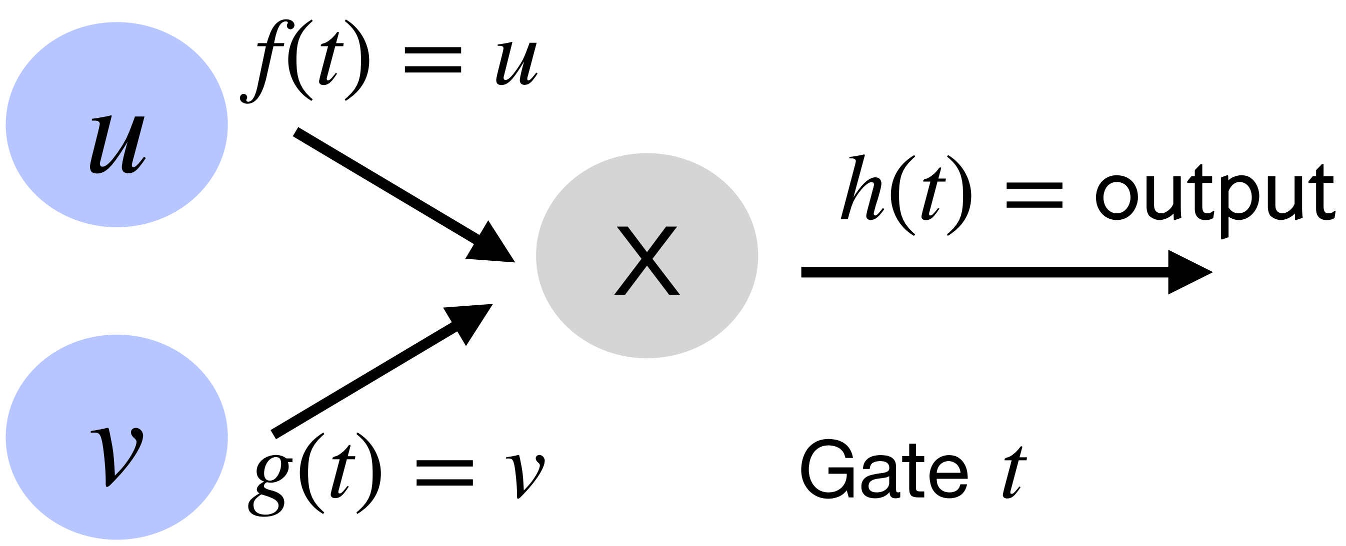
Server verification

1. Server can reconstruct secret shares for every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g
3. Check that h is well-formed $(f \cdot g = h)$
   - Servers choose random r
   - Evaluate $\alpha \leftarrow f(r) \cdot g(r) - h(r)$
   - If $f \cdot g \neq h$, $\Pr[\alpha = 0]$ is negligible
   - Checking this requires 1 multiplication of secret-shared values

How do SNIPs work?

Client proof generation

1. Define 3 polynomials: f, g, h
2. For multiplication gate t with inputs u_t, v_t:
   - $f(t) = u_t$
   - $g(t) = v_t$



3. Let $h = f \cdot g$
4. Secret-share coefficients of h to 2 servers

Server verification

1. Server can reconstruct secret shares of every wire value
   - Has share of input
   - Has share of each multiplication output
   - Can derive outputs of addition gates
2. Servers can construct shares of polynomials f, g
3. Check that h is well-formed $(f \cdot g = h)$
   - Servers choose random r
   - Evaluate $\alpha \leftarrow f(r) \cdot g(r) - h(r)$
   - If $f \cdot g \neq h$, $\Pr[\alpha = 0]$ is negligible
   - Checking this requires 1 multiplication of secret-shared values
4. Check output wire is shares of 1 to verify $\text{Valid}(x) = 1$

What can Prio compute with sums?

If you can compute private sums, you can compute many other interesting aggregates using known techniques

- Average
- Variance
- Standard deviation
- Most popular (approx)
- Frequency counts for a small set
- Min and max (approx)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

Outline

1. Prio design
2. **Tim Geoghegan and Chris Patton**

References

Bonawitz, Keith, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical secure aggregation for federated learning on user-held data." *arXiv preprint arXiv:1611.04482* (2016).

Corrigan-Gibbs, Henry, and Dan Boneh. "Prio: Private, robust, and scalable computation of aggregate statistics." In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pp. 259-282. 2017.

Prio slides from Henry Corrigan-Gibbs: https://people.csail.mit.edu/henrycg/files/academic/pres/nsdi17prio-slides.pdf