Applications of MPC

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 14 (March 19, 2025)

> ***Warning:*** This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Background: MPC in practice
- Client-server model
- Private aggregation
- Security against malicious clients

Introduction

In the last lecture, we discussed the BGW protocol for secure multiparty computation (MPC). The BGW protocol is powerful and flexible: it gives a secure multiparty protocol for computing *every* efficiently computable function.

Today, we will shift our focus to applications of MPC protocols. The questions we will consider are:

- What functions do companies/businesses today actually want to compute in a multiparty fashion?
- What are the limitations of MPC protocols that make them hard to use in practice?
- How do large-scale deployments of MPC technology get around these barriers?

Note on threat model: When you see a practical application of MPC technology it is worth thinking carefully about what types of attack the system can and cannot defend against.

As one concrete example, MPC technology does not by nature give any protection against software bugs: in most implementations of n-party MPC protocols, all n parties run exactly the same MPC software. So an attacker that finds an over-the-network-exploitable bug in one party's code can compromise all n parties in one go—compromising n machines in this case almost as easy as compromising one.

What MPC protocols *can* help with is local compromise: if two companies run an MPC, an MPC can protect one company from learning more than it should about the other's data.

Applications of MPC

Let us start by discussing applications of MPC in deployed systems today. As far as I know, there are three major applications of MPC today:

Application: Custody of cryptocurrency secrets (many blockchain companies). Say that you run an organization that holds cryptocurrency assets. There is a secret signing key—or often, many keys—associated with these assets. Anyone holding the secret key can authorize transactions to transfer these assets to other accounts.

Given that the key could be worth millions of dollars, you may not trust any one employee to hold the secret key. Even if you had such a trustworthy employee, you not want to give that employee the sole responsibility of safeguarding the key. (You can think about why that might be.)

So, a common way to manage secrets is to split the signing key using a secret-sharing scheme, and to have different employees manage each key share. Whenever the organization wants to authorize a transaction, the employees jointly run a computation to generate a signature on the transaction.

Rather than using BGW or some other general-purpose MPC protocol, most companies implementing these signing protocols use special-purpose "threshold signing" protocols. These are often more complicated and less elegant than BGW, but they are much much faster.

There are scores of companies offering this "MPC wallet" service to enterprise customers.

Attackers still manage to compromise MPC wallets; one reason is, as I mentioned above, MPC does not protect against software bugs. The threshold-signing schemes that many wallet systems implement are quite subtle, complicated, and easy to get wrong. So it is no surprise that bugs abound.

Application: Private JOIN for ad attribution (Google, Meta). Google and Meta both apparently use MPC for "conversion measurement" in online ads. In that setting, Google (for example), holds a list of users to whom it showed a BMW ad, e.g., listed by phone number. BMW, holds a list of users who bought a car, also with the purchaser's phone number. The two parties want to know: how many people who saw the BMW ad bought a car?

At the same time:

- Google wants to reveal as little as possible about its ad data to BMW and

- BMW wants to reveal as little as possible about its customer data to Google.

Apparently—and I say apparently because this is very difficult for me to verify—Google and Meta use MPC to solve this problem.

Application: Private aggregation (Apple, Google, Mozilla, etc.). The final application, "private aggregation," is one that is near and dear to my heart, since I have been working on these systems for many years now.

In this application, a company has millions of clients, where each client i holds a value x_i. The company wants to compute some function $f(x_1, x_2, \dots)$, over all of the x_is, without learning anything more about any individual value x_i.

This problem comes up in private-telemetry applications. For example, Apple wants to know what things iPhone users usually photograph in Paris. (Apple uses this data to produce "memories" reels in the Photos app.)

Apple, Mozilla, and other companies use a certain type of MPC protocol to solve this type of private-telemetry problem.

Concrete efficiency of multiparty computation

So far, we have looked at secure multiparty computation from a theoretical perspective. In the BGW protocol, for example:

1. we model a computation as an arithmetic circuit (i.e., a circuit with $+$ and $\times$ gates modulo p) and

2. we treat all parties as having point-to-point secret and authenticated communication channels.

In practice, each of these two modeling features can be problematic:

1. Many computations do not naturally have "nice" representations as arithmetic circuits. For example, a simple RAM program (e.g., a Python script) can compile into an arithmetic circuit with trillions of gates.

In the BGW protocol, the communication cost scales linearly with the number of gates in the circuit (number of multiplication gates, to be precise). The number of rounds of communication scales linearly with the depth of the circuit.

So big circuits yield slow implementations.

2. If the number of parties is in the millions, it may be infeasible for them all to have point-to-point communication links with each other. It would in principle be possible to proxy all of the party-to-party communication through a central server, but in a dynamic environment such as the Internet, it may be difficult to even nail down who the parties *are*, much less have them all agree on shared secrets with each other.

So, if we want to use secure multiparty computation in practice, what are we to do? We need to cheat in two different ways.

Avoid large circuits: Focus on interesting special cases.

Rather than use a BGW-like (i.e., general-purpose) MPC protocol directly, applications will typically use a special-purpose MPC protocol. This sidesteps the issue that most interesting computations do not have small circuit representations.

Let's discuss each of the three applications:

- For the MPC-wallet and private-join applications, companies typically use an ad-hoc special-purpose MPC protocol that looks nothing like BGW. These protocols very carefully exploit the structure of the computation that the parties are trying to run in an MPC to get savings.

- For the private-telemetry application, companies use a protocol that looks very similar to BGW. The only twist is that they only run the protocol with circuits that have addition gates (*no* multiplication gates!).

  The simplest example of such a circuit is the private-sum circuit, which takes as input a value x_i from each client i, and outputs the sum $\sum_i x_i$. Even though this functionality seems trivial (and it is!), it is already useful enough for a number of practical applications. In fact, the largest-scale MPC protocols ever run implement this private-sum protocol.

As far as I know, all applications of MPC today use one of these two approaches. I know of no MPC protocol in use today that, say, implements training of a neural net using BGW. The communication/computation cost of applying general-purpose MPC in such settings is just too large.

Avoid client-to-client interaction.

The other issue we have in BGW is that we assumed that there is a fixed/static group of parties to the protocol.

- In the private-join application, there are only two parties, and the parties have a pre-existing business relationship. So participant churn is not an issue.
- In the MPC wallet application, there are typically only a small number of parties. It is possible to make MPC protocols robust to the failure of a fraction of the parties, and implementations use these tricks to allow making progress even if some of the parties have failed.
- In the private-telemetry application, there are millions of parties, so churn is a concern.

  This performance cheat gets rid of the need for client-to-client communication entirely. This is convenient because the set of clients may evolve quickly over time (before even a single iteration of the multiparty protocol has completed) and also because communication is costly.

  A common technique to handle participant churn here is to work in the *client/server model*. The idea is to shift most of the work of running the multiparty computation onto a small number of powerful parties ("servers")—maybe only two or three. Each client secret-shares their input to the $k \ll n$ servers, who run the multiparty computation amongst themselves and return the answer to the clients.

  This approach has major benefits in terms of communication cost: Each client only communicates with the k servers and each client's communication cost is *independent* of the number of clients.

  Compare this per-client cost with that of BGW.

  In addition, we only need to worry about keeping the $k \ll n$ servers online in steady state. To participate in the computation, each client only needs to stay online long enough to submit shares of its input to the servers.

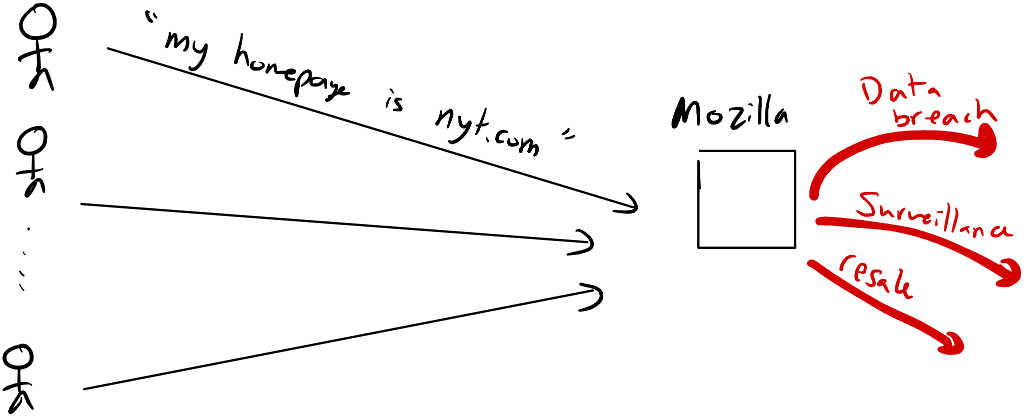
  For example, Apple runs one server and a second organization runs the other server. Divvi Up is a non-profit whose only job is to run the second-server for private-telemetry MPC protocols.

  The downside of this approach is security: An attacker now only needs to compromise $k \ll n$ servers to violate the privacy of (i.e., recover the secret inputs of) all clients. For example, if $k = 2$, then if the two servers decide to get together and share their information, they learn the private data of all clients.

Motivating application: Private analytics

Example: Mozilla (co behind firefox web browser) wants to know: Which webpages in the Alexa top 10k are popular as homepages?

Non-private analytics



Problem: Mozilla learns every person's home page. → Privacy failure.

→ Goal: Compute the aggregate statistic of interest (which pages are popular) while learning nothing else.

Lots of use cases: ("Prio" w/ Dan Boneh 2017)

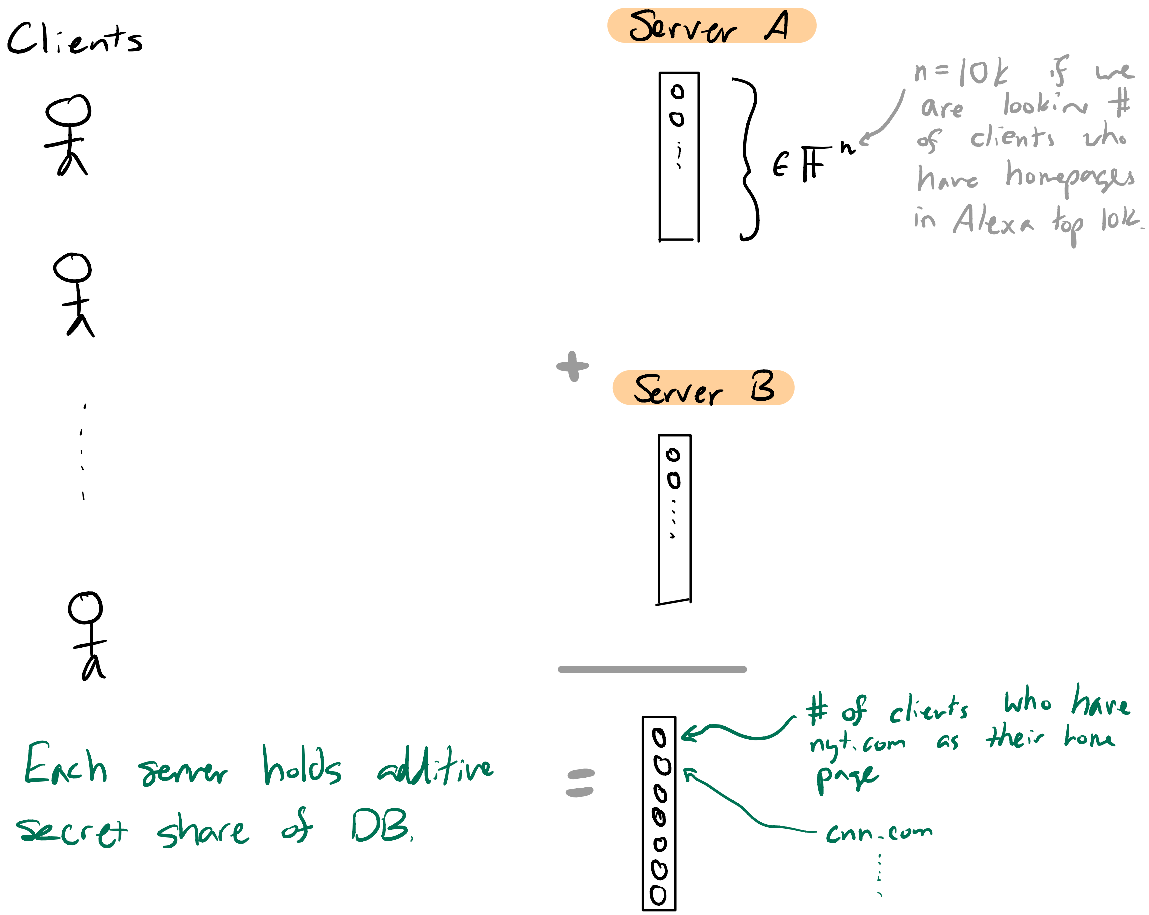
- * Apple uses this stuff to gather data on usage of the Photos app.
- * Mozilla uses this technology for gathering browser-usage data
- * Apple, Google used these techniques for gathering stats on Covid EN apps
- * Online-ad cos are working on using this tech to gather ad-conversion data....

Private analytics (Private Histograms)

Idea:
- Have 2+ servers store database in secret-shared form.
- Clients can "write into DB" w/o revealing where/what they wrote.

Throughout, all arithmetic will be in finite field $\mathbb{F}$

(Think: integers mod ~80-bit prime)



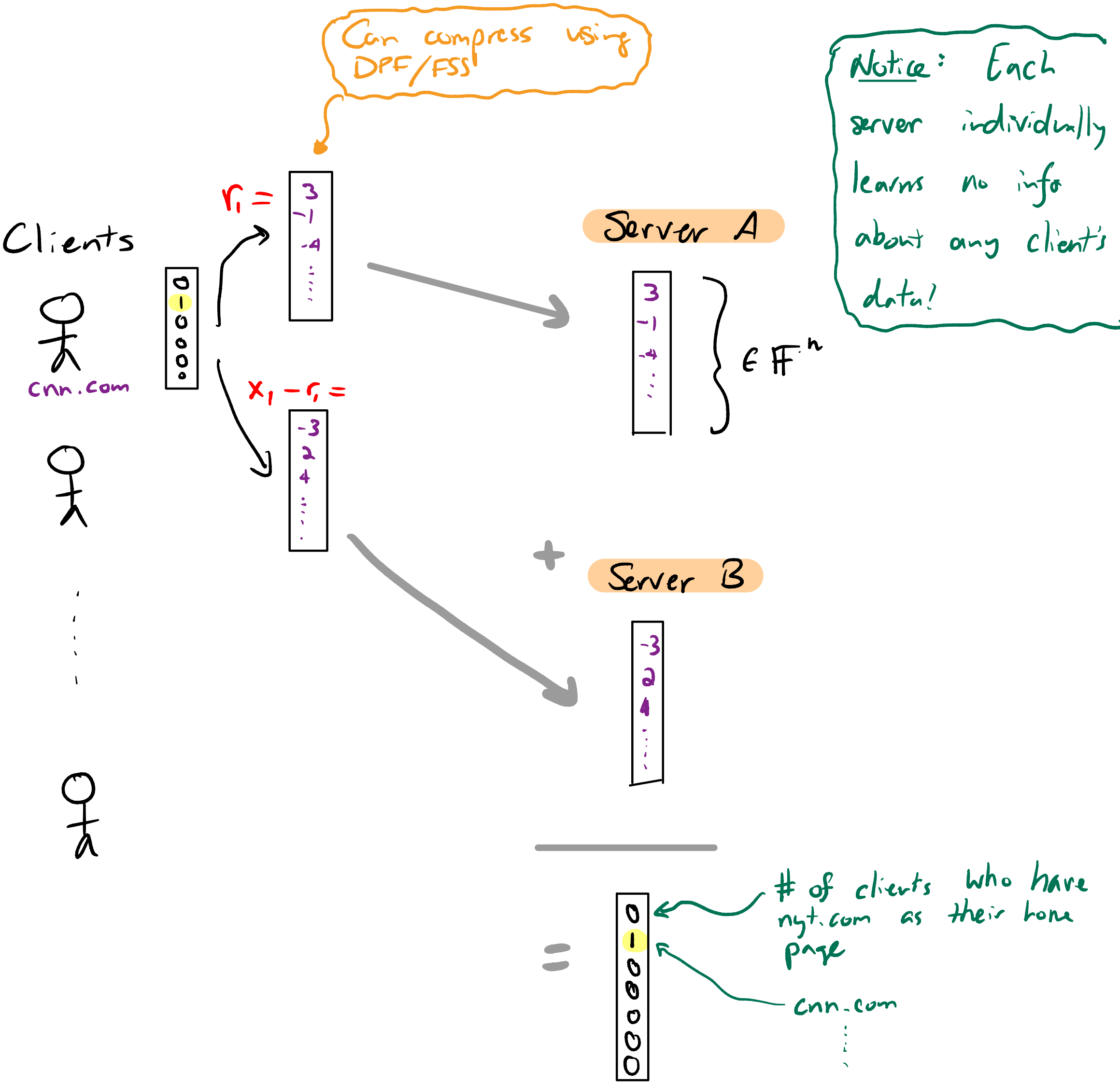
n = 10k if we are lookin # of clients who have homepages in Alexa top 10k.

# of clients who have nyt.com as their home page

cnn.com

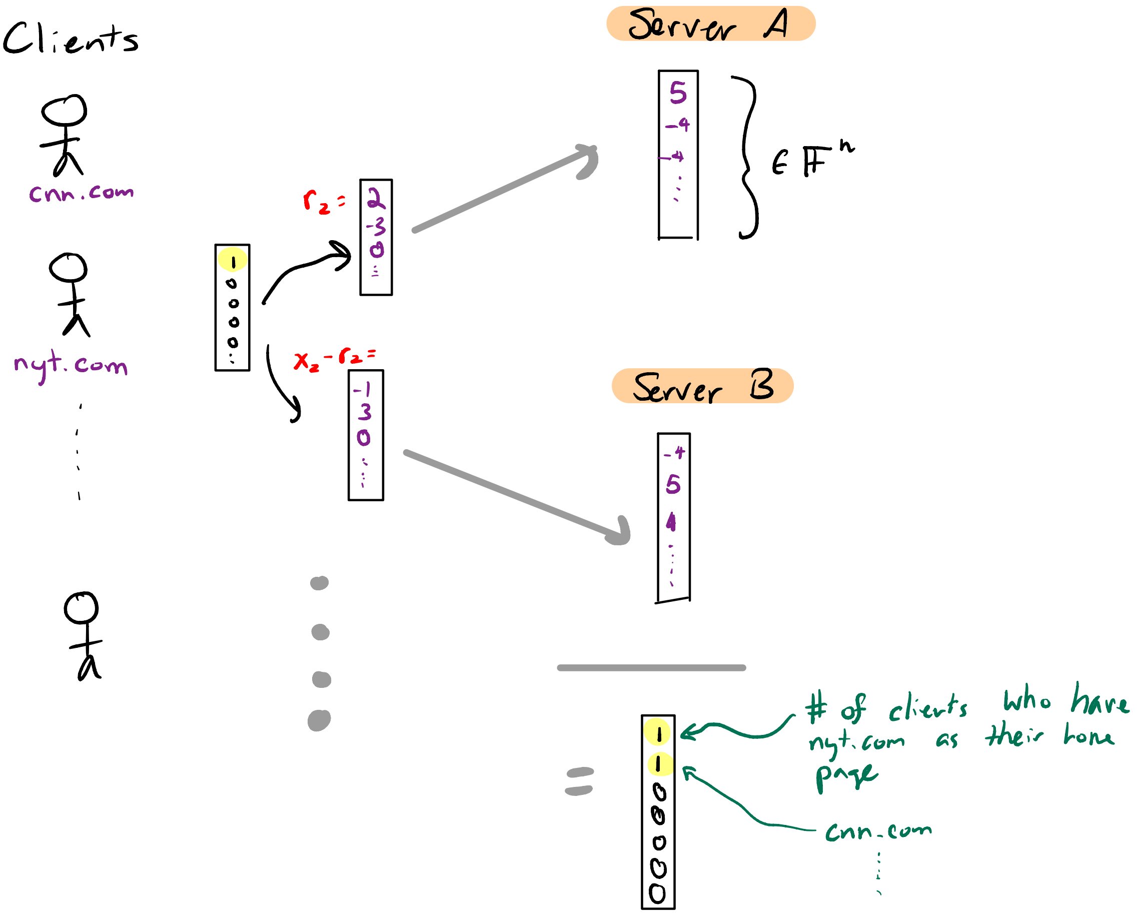
Each server holds additive secret share of DB.

- Clients send secret-shared DB "updates"
- Servers learn nothing* about which client has which homepage.

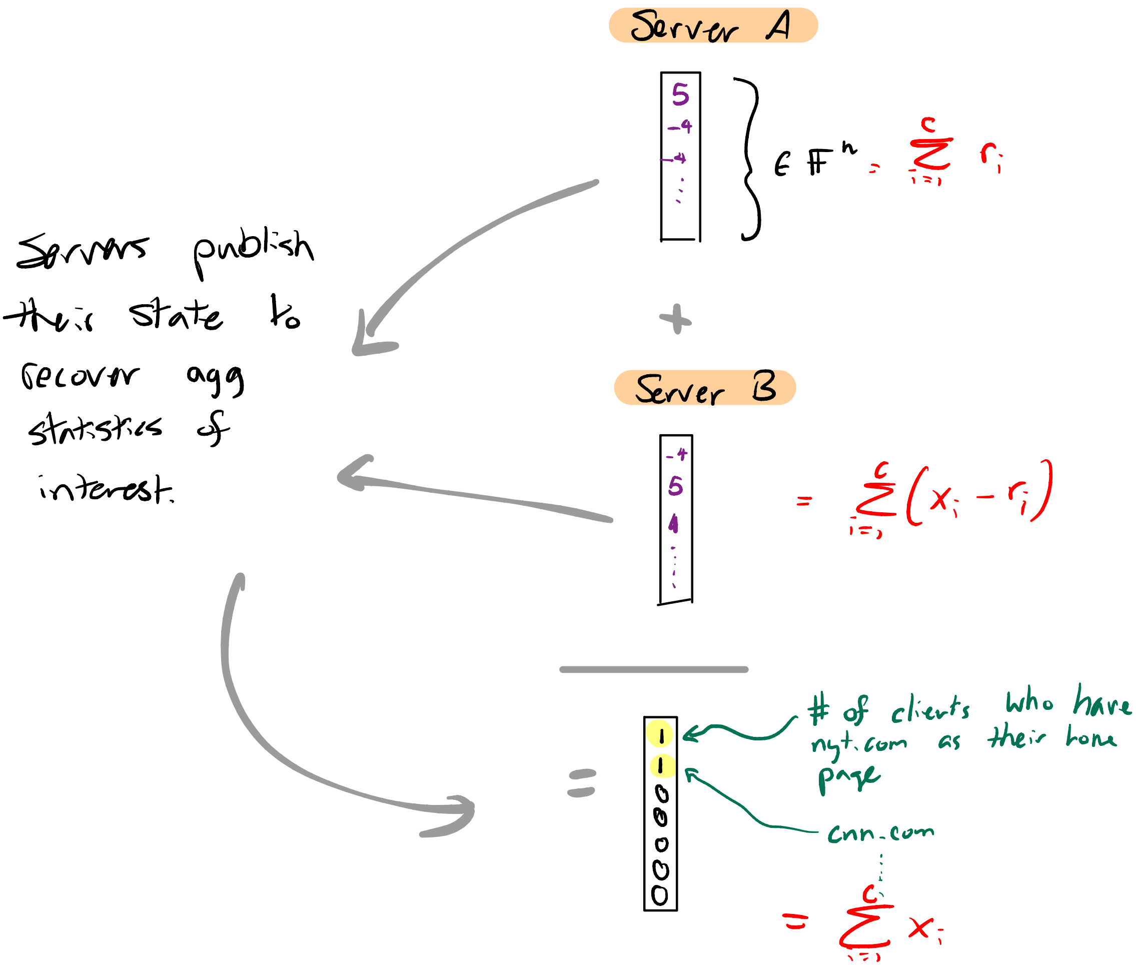


Notice: Each server individually learns no info about any client's data!

Each client sends one of these "updates"....



At the end of the day (or reporting period),



Server A

$\in \mathbb{F}^n = \sum_{i=1}^{c} r_i$

+

Server B

$= \sum_{i=1}^{c} (x_i - r_i)$

Servers publish their state to recover agg statistics of interest.

=

# of clients who have nyt.com as their home page

cnn.com

$= \sum_{i=1}^{c} x_i$

Private Analytics so far

This simple scheme is already useful!

0. Simple, easy to implement, concretely efficient (as long as # websites/counters not too big)

1. Correct.

   Clients and servers execute protocol
   ⇒ Servers get correct output.

   [For each top-10K website w, # of clients that have w as a homepage.]

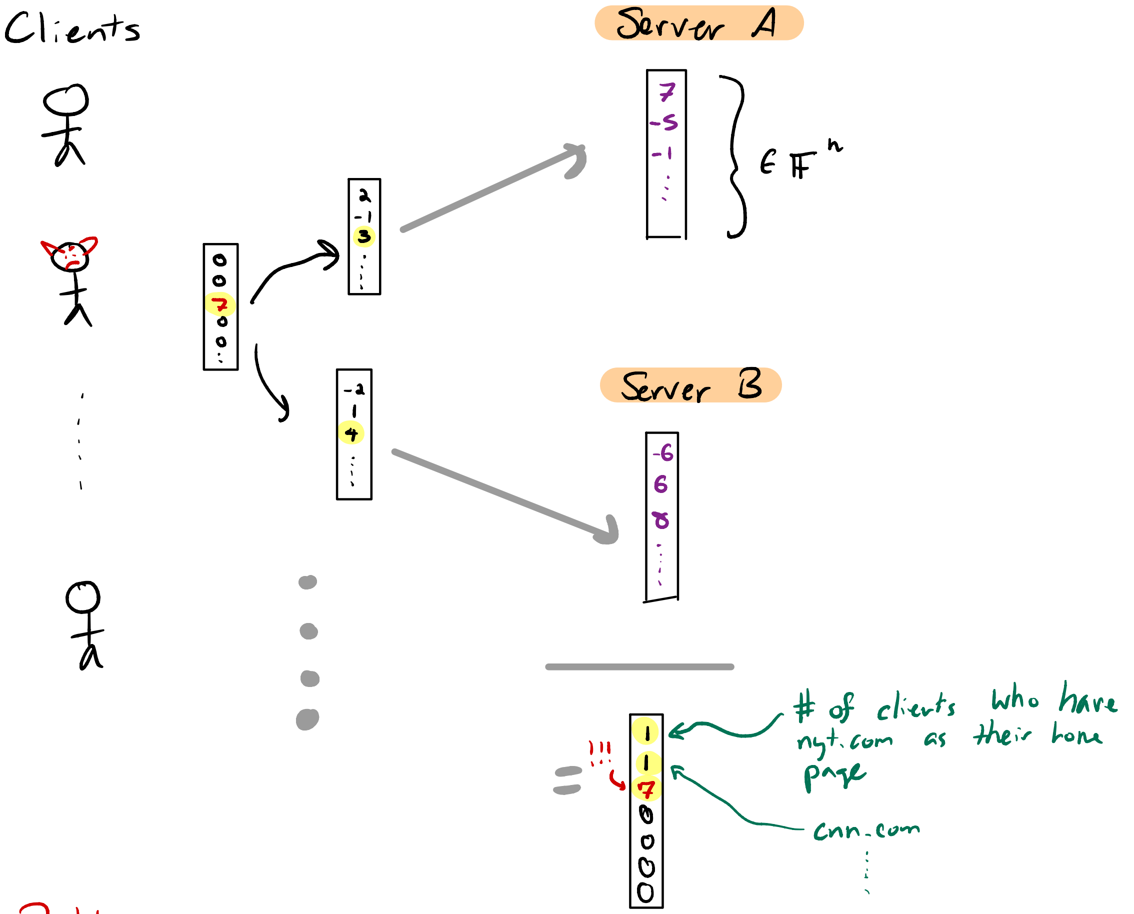
2. Private.

   Malicious server "learns nothing" about client data except what it can infer from the protocol output.

   $\forall$ inputs $x_1, \ldots, x_c$ $\forall$ adv A, $\exists$ Sim s.t.

   $$\left\{ \begin{array}{l} \text{View of mal} \\ \text{server in} \\ \text{execution with} \\ \text{client inputs} \\ x_1, \ldots, x_c \end{array} \right\} \approx \left\{ \mathrm{Sim}(f(x_1, \ldots, x_c)) \right\}$$

Problem: One malicious client can completely mess up the computation!



Problems

* One client can "vote" many times
* Client can submit negative votes
* Client can cause output to be random garbage

→ Even worse, servers can't tell when this happens or who to blame!

Rest of hour: How to defend against misbehavior in simple secret-sharing-based protocols.

↳ Without paying "too much" in concrete efficiency.

BEWARE!

When trying to achieve malicious security, everything gets complicated.

* Malicious clients?
  servers?
  clients and servers?

* Care about advs corrupting output? Or only breaking client privacy?

.... malicious security
what keeps me up at night ☹

e.g. BGW

Servers can use general-purpose MPC ^ to detect & prevent client misbehavior.

↳ $\Omega(n)$ communication per client

Better Idea: Special-Purpose Scheme

↳ $O(1)$ # elms of comm per client

Simplest example:

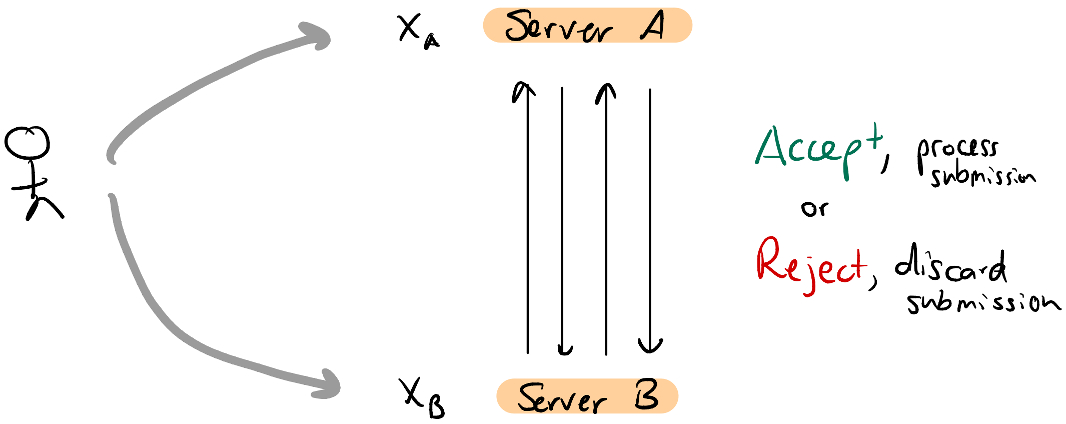
- * Protecting the private-analytics scheme from malicious clients.
- * Ensure that each client can only "vote once"

Idea [BGI '16]: Have servers check that each client submission is well-formed before accepting it.

↳ If not, reject/ignore client submission.

* Honest client will send shares $X_A, X_B \in \mathbb{F}^n$ whose sum is an all-zero vector with a single 1.

* Servers want to check this property without (a) communicating to much
(b) breaking client privacy.



Sketch-verification scheme

Servers hold additive shares of $x \in \mathbb{F}^n$

Let $\mathcal{L} = \left\{ \begin{matrix} \text{vectors of all 0's} \\ \text{w/ at most 1 one} \end{matrix} \right\} \subseteq \mathbb{F}^n$

Completeness

$x \in \mathcal{L} \Rightarrow$ honest servers accept

Soundness

$x \notin \mathcal{L} \Rightarrow$ Honest servers reject w.h.p.

Privacy/HVZK

$x \in \mathcal{L} \Rightarrow$ Honest server "learns nothing else" about x.

$\exists$ Simulator S $\forall$ $x_a, x_B \in \mathbb{F}^n$ s.t. $x_A + x_B \in \mathcal{L}$

$$\left\{ \begin{matrix} \text{View of} \\ \text{server } A \\ \text{on inputs } (x_A, x_B) \end{matrix} \right\} = \left\{ \text{Sim}(\;) \right\}$$

Sketch-verification [BGI'16]

- Servers use a linear function (sketch) to compress vector in $\mathbb{F}^n \to \mathbb{F}^{O(1)}$

  ↳ Sketch output summarizes goodness/badness of input.

  Key: Computing linear fns on additively sec shared data requires no interaction.

- Servers use $O(1)$-size MPC to check sketch values

[This sketching idea applies broadly to secret-sharing based protocols... it's a useful trick to know.]

Protocol [Boyle, Gilboa, Ishai '16]

For checking $x \in \mathcal{L} = \left\{ \begin{array}{l} \text{vectors of all zeros} \\ \text{w/} \leq 1 \text{ one} \end{array} \right\} \subseteq \mathbb{F}^n$:

1. Servers agree on random values (unknown to client)

$$r = (r_1, \dots, r_n) \in \mathbb{F}^n$$
$$R = (r_1^2, \dots, r_n^2) \in \mathbb{F}^n$$

Inner product

$$\sum_{i=1}^{n} (x_B)_i r_i \in \mathbb{F}$$

2. Compute "test" values

Server A

$x_A \in \mathbb{Z}_p^n$

$t_A = \langle x_A, r \rangle \in \mathbb{F}$

$T_A = \langle x_A, R \rangle \in \mathbb{F}$

Server B

$x_B \in \mathbb{F}^n$

$t_B = \overbrace{\langle x_B, r \rangle} \in \mathbb{F}$] Shares of $\langle x, r \rangle$

$T_B = \langle x_B, R \rangle \in \mathbb{F}$] Shares of $\langle x, R \rangle$

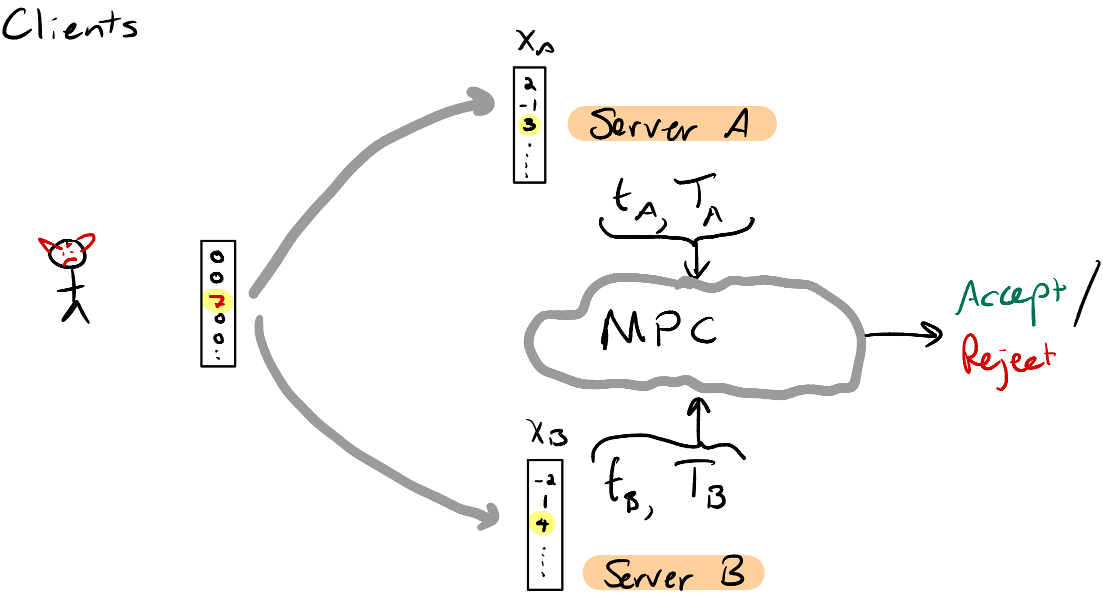
3. Use $O(1)$-sized MPC to check that

$$(t_A + t_B)^2 - (T_A + T_B) = 0 \quad \in \mathbb{F}$$

Communication: $O(1)$ $\mathbb{F}$ elements — indep of n!

Computation: $O(n)$ ops in $\mathbb{F}$

With sketch-verification, malicious clients can no longer mess up the computation.



Analysis.

Completeness
($x \in \mathcal{L} \Rightarrow$ Honest servers accept)

Servers effectively test whether:

$$\langle x, r \rangle^2 - \langle x, R \rangle \overset{?}{=} 0$$

If x has ≤ 1 non-zero coordinate

$$r_i^2 - r_i^2 = 0 \quad ✓$$

Soundness

- Appeal to Schwartz-Zippel Lemma
- Cheating client wins w.p. $\leq \frac{2}{p}$.

Privacy / HVZK

- Servers only learn single-bit output of MPC.
- If $x \in \mathcal{L}$, can simulate trivially.

Beware: Selective-failure attacks

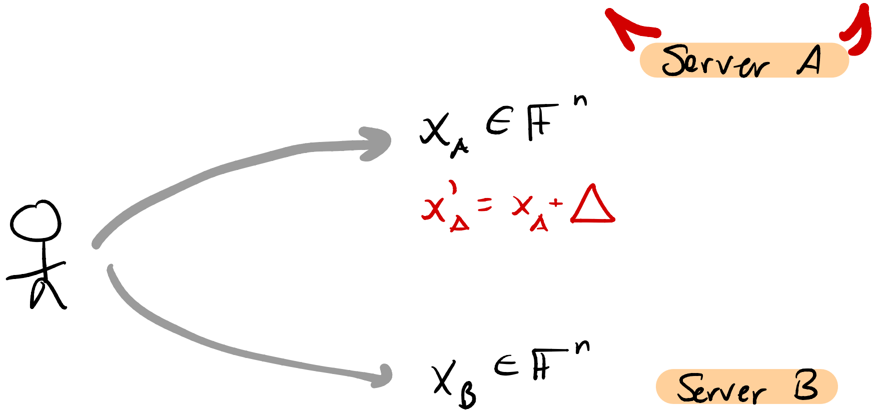
The simple sketching scheme does not protect client privacy/ZK against a malicious server.

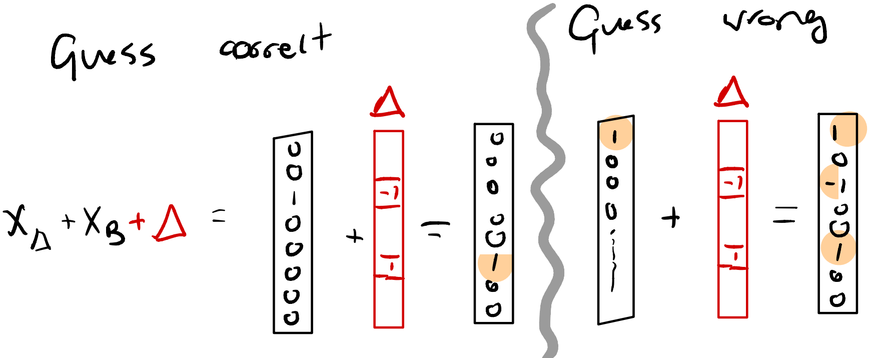
↳ Only HVZK

Attack:
- * Malicious server guesses location of non-zero element and shifts it
- * Rest of protocol is the same.
- * Guess correct → Verif accepts

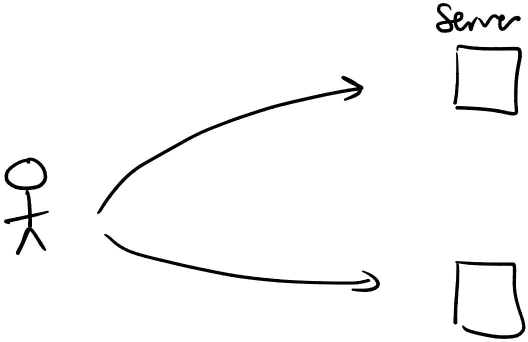
  Guess wrong → Verif rejects

Acc/rej bit leaks client data.





Summing up: sketching can check "simple" predicates with little communication.



Using sketching, in the 2+-server setting can construct private analytics system that protects against malicious clients.

For more complicated predicates, sketching is insufficient e.g. check that $x_A + x_B \in \{0,1\}^n \subseteq \mathbb{F}^n$.

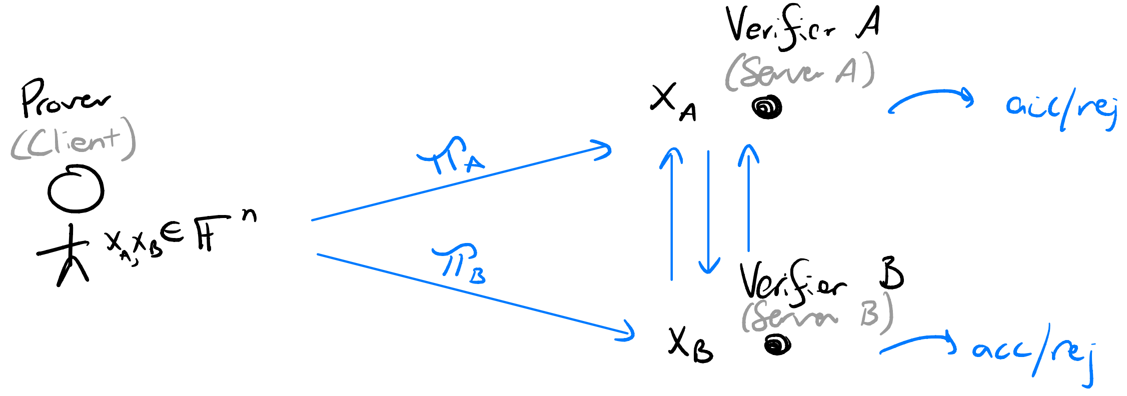
[In some cases, can even prove that checking such predicates on secret-shared data requires a lot of communication. Uses comm complexity args,]

What do we do then???

ZK Proofs on Secret-Shared Data

w/ Boneh, Boyle, Gilboa, Ishai

"Sketch-verification with a proof"



Prover's goal: Convince verifiers that $x_A + x_B \in \mathcal{L}$, for some $\mathcal{L} \subseteq \mathbb{F}^n$

As in sketch verification schemes. want

Completeness $\forall\ x_A + x_B \in \mathcal{L}$

$$\Pr[\langle P, V_A, V_B \rangle (x_A, x_B) = \text{"acc"}] = 1$$

Soundness $\forall\ x_A + x_B \notin \mathcal{L},\ \forall\ P^*$

$$\Pr[\langle P^*, V_A, V_B \rangle (x_A, x_B) = \text{"acc"}] \leq \text{small}$$

Privacy / HVZK $\exists$ Sim s.t. $\forall\ x_A + x_B \in \mathcal{L}$

$$\left\{ \begin{matrix} \text{View of server } A \\ \text{in } \langle P, V_A, V_B \rangle (x_A, x_B) \end{matrix} \right\} = \{ \text{Sim}() \}$$

Useful results to know

Let $\mathcal{L} \subseteq \mathbb{F}^n$ be a language

Thm (BBCGI'19)
If $\mathcal{L}$ is computable by an arithmetic ckt of size $|C|$, there is a ZK proof on secret-shared data for $\mathcal{L}$ with comm cost P→V $O(|C|)$, V→V $O(1)$ elements of $\mathbb{F}$.

If P and Vs can interact, cost falls to $O(\log|C|)$ sublinear in certain special cases – when C has structure. (e.g. C has degree two)

Construction proving Thm is info-theoretic & simple
⇒ Very general tool for protecting against misbehavior in secret-sharing based protocols.

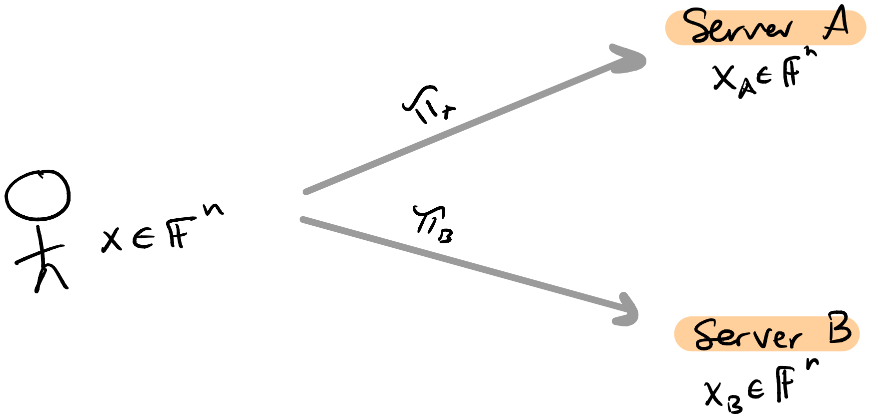
* Sublinear variant is especially useful for getting malicious security at $o(1)$ overhead.

The full result uses very nice abstractions (linear PCPs) that I will not have time to describe.

Instead, will give you one special case that gives a flavor of the result.

Example: Checking that a secret-shared vector is in $\{0,1\}^n \subseteq \mathbb{F}^n$

Application: Mozilla wants to know which websites in Alexa top 10k use Flash (or other browser feature) w/o learning who is visiting which site.



Servers want to check $x_A + x_B \in \{0,1\}^n \subseteq \mathbb{F}^n$

Idea: Let $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$

Define polynomials f, g, h over $\mathbb{F}$ s.t. $\forall i \in [n]$

$$\left.\begin{aligned} f(i) &= x_i \\ g(i) &= x_i - 1 \end{aligned}\right\} \deg \leq n-1$$

$$h = f \cdot g \quad \Big\} \deg \leq 2n-1$$

Abusing notation here... associating $1, \ldots, n$ with distinct elms of $\mathbb{F}$

If $x \in \mathcal{L}$, then $\forall i \in [n]$ $h(i) = 0 \in \mathbb{F}$

$h(i) = f(i) \cdot g(i) = x_i (x_i - 1) = 0$ when $x_i \in \{0,1\}$

If $x \notin \mathcal{L}$ then $\exists$ $i \in [n]$ s.t. $h(i) \neq 0 \in \mathbb{F}$

By same argument.

Key facts: 1. Given shares of x, can compute shares of $f(r)$, $g(r)$ for $r \in \mathbb{F}$ w/o comm.

2. Given shares of coeff of h, can compute shares of $h(r)$ for $r \in \mathbb{F}$ w/o comm.

Recipe (Simplified!)

1. Client computes poly h. Sends shares of h to servers as proof π. $\approx n$ $\mathbb{F}$ elements

2. Servers check that $\forall i \in [n]$ $h(i) = 0$.
   ↳ Servers compute shares of $h(1), \dots, h(n)$
   ↳ Publish random lin comb of shares, check $= 0$

3. Servers check that $f \cdot g = h$ (client constructed pf well)
   ↳ Servers choose random $r \xleftarrow{R} \mathbb{F}$.
   ↳ Compute shares of $f(r), g(r), h(r)$
   ↳ Use $O(1)$-sized MPC to check that
   $f(r) \cdot g(r) - h(r) = 0$.

With a few extra tweaks, don't even need MPC step here.

$P \to V$ comm: $\approx n$ $\mathbb{F}$ elms each

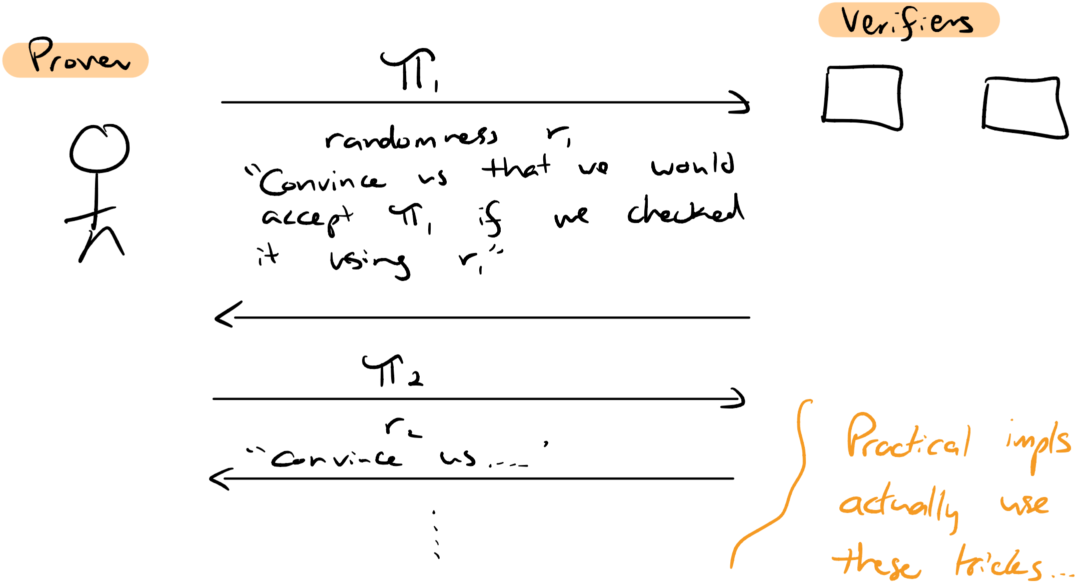
$V \to V$ comm: ≈ 4 $\mathbb{F}$ elms each

Simple! Info-theoretic!

(This idea generalizes naturally to handle langs computed by arbitrary circuits over $\mathbb{F}$.)

- When ckt computing $\mathcal{L}$ is structured, we can reduce the $P \to V$ comm (pf size).

- Basic idea:
  - * Say that lang $\mathcal{L}_n$ recognized by circuit C_n consisting of n repeated subcircuits.
    (e.g. checking $x \in \{0,1\}^n \subseteq \mathbb{F}^n$)
  - * Construct ZK pf on sec-shared data s.t. verifiers need to run an MPC of $C_{n/2}$ to check the proof
  - * Recursively use ZK pf....


Prover
Verifiers
π_1
randomness r_1
"Convince us that we would accept π_1 if we checked it using r_1"
π_2
r_2
"convince us...."

Practical impls actually use these tricks...

Summing up...

Two tools for protecting against misbehavior in protocols using secret sharing.

1. Sketch-verification
   - * For testing simple predicates on large vectors w/ small comm.
   - * No help from client needed

2. ZK Proofs on Secret-Shared Data
   - * For testing arbitrary predicates on secret-shared data
   - * Requires help from client
   - * Comm depends on ckt complexity of predicate
     - ↳ Can be sublinear in ckt size.