Private Aggregation & Proofs on Secret-Shared Data

Today

* Recap: SNARGs & Linear PCPs
* Private aggregation
* Simple scheme & its problems
* Fix: Proofs on secret-shared data
  ↳ Fully linear PCPs

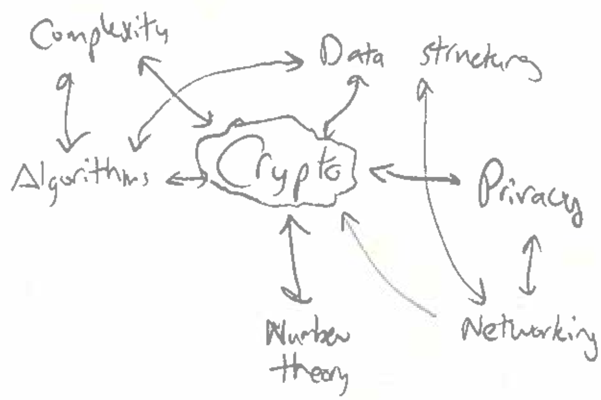
(Story)

Logistics

- Problem Set 5 is out! Due 6/8 at 5pm on Gradescope
- OHs today — David's OH moved to 2:30pm today
- Research...
- David's defense

Today, we are covering results that are "hot off the press"

- Application of very recent techniques to privacy problem
  ↳ Browser vendor computing the most popular homepage w/o learning anything else
- Fancy crypto not just for making $! ☺ Also for protecting privacy.
- Why crypto is awesome!... from theory theory to practice in one lecture!



- This is not only theoretical...

Recap: e.g. Graph 3-coloring:

Normal NP proof: $P \xrightarrow{\text{3-colorin of } G,\ \Omega(|V|) \text{ bits}} V$

SNARG $P \xrightarrow{\text{SNARG pf},\ 8\lambda \text{ bits}} V$

No matter how large graph is!

* Proof shorter than NP witness!

* Good evidence that SNARGs don't exist for all NP lang under "standard" assumptions

what is this?

We constructed SNARG using general strategy (BCIOP, IKO, ...)

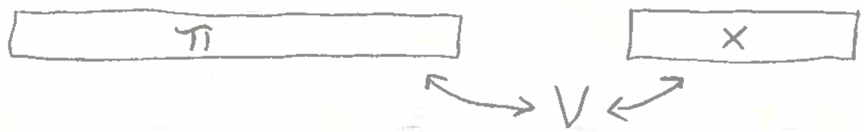
$\underbrace{\text{Linear PCP for } \mathcal{L}}_{\substack{\text{Info-theoretic part} \\ \text{(no assumptions)}}} \underbrace{\xrightarrow{\text{Crypto compiler using "linear-only" encryption}}}_{\substack{\text{Crypto part} \\ \text{(uses assumptions)}}} \text{SNARG for } \mathcal{L}$

Since we will be using Linear PCPs again today, want to refresh your memory.

Types of Proof (info theoretic / no assumptions)   Language $\mathcal{L} \subseteq \{0,1\}^*$

$$\underline{P(x)} \xrightarrow{\quad \pi \quad} \underline{V(x)} \to \text{acc/rej}$$

Generally, think of V as being "given access" to π and x.



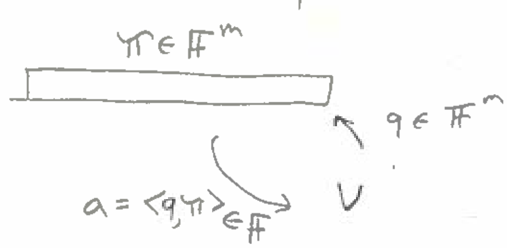
(For this lecture, think of π as poly size)

Proof Type	Access to x	Access to π
NP/MA	Read all	Read all
PCP	Read all	Point query (Read bits)
Linear PCP	Read all	Linear query
Fully linear PCP	Linear query	Linear query ← Today
PCPP	Point query	Point query
⋮		

Many more! Also interactive!

Point query

Linear query

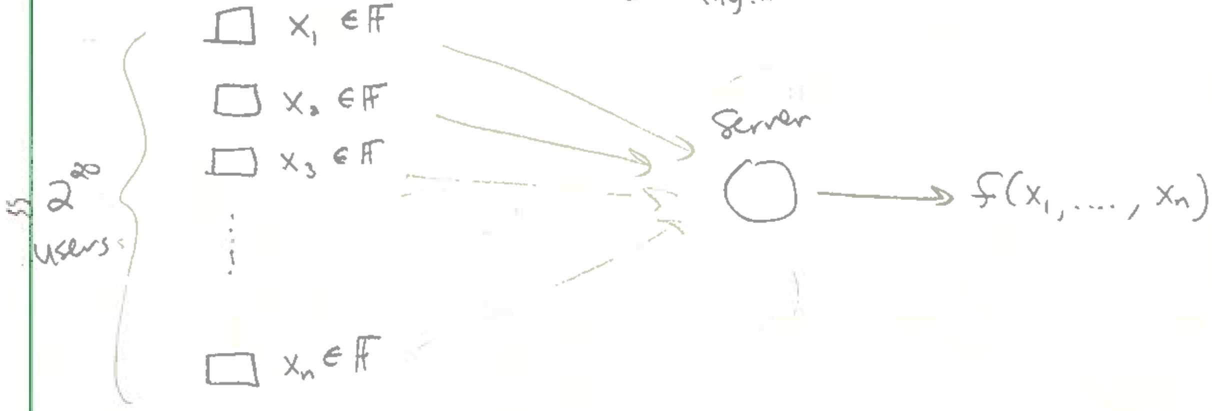


In a fully linear PCP, the verifier has restricted access to the input and the proof π.

Q: Can you construct a FLPCP for $\mathcal{L}$ from an NP proof for $\mathcal{L}$?
PCP " NP proof " ?
MA proof " LPCP proof " ?

Private Aggregation

Ideally....



Let $f: \mathbb{F}^n \to \mathbb{F}$ be a function

Problem: Want to compute $f(x_1, \ldots, x_n)$ without revealing "anything else" about $x_1, \ldots, x_n$ to server.

→ When would this be useful?

E.g. x_i is speed of car i on Bay Bridge

$f(\,)$ computes average speed

↳ Learn avg speed w/o leaking any individual's speed

E.g. x_i is 0/1 value: Browser i has Stanford.edu as its homepage.

$f(\cdot)$ computes sum of x_i

↳ Learn how many people use stanford.edu as homepage w/o leaking anything else.

E.g. x_i is location of phone i

$f(\cdot)$ computes most popular value amongst inputs

↳ Learn pop location w/o leaking any individuals location

Two general approaches

1) Local differential privacy

2) MPC-based ← Today

We'll simplify the problem a bit

1) We'll use 2+ "non-colluding" servers → For practical reasons... makes easy to handle clients that fail

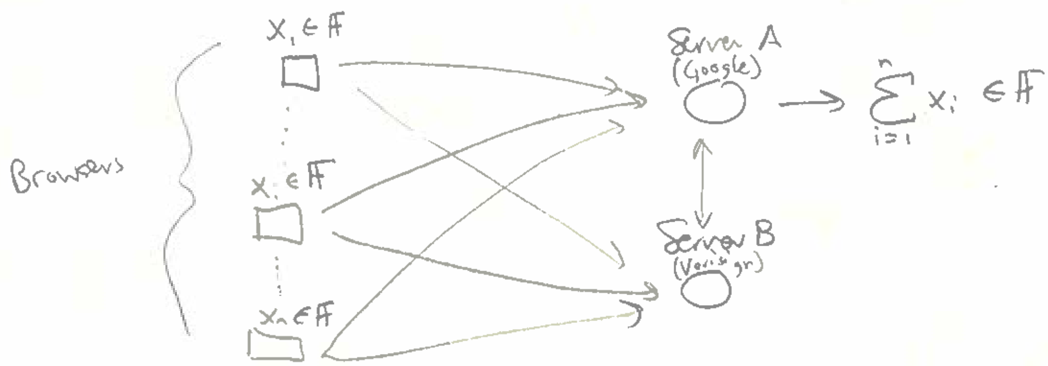
2) We'll focus on "simple" functions f
↳ to avoid general MPC

Consider 2-server case (generalizes easily to many servers)

Problem Statement

Each client i holds $x_i \in \mathbb{F}$ (e.g. 0/1 value saying whether homepage is stanford.edu)

Servers want to compute $f(x_1, \dots, x_n) = \sum_{i=1}^{n} x_i \in \mathbb{F}$ (Popularity of Stanford.edu as home page)

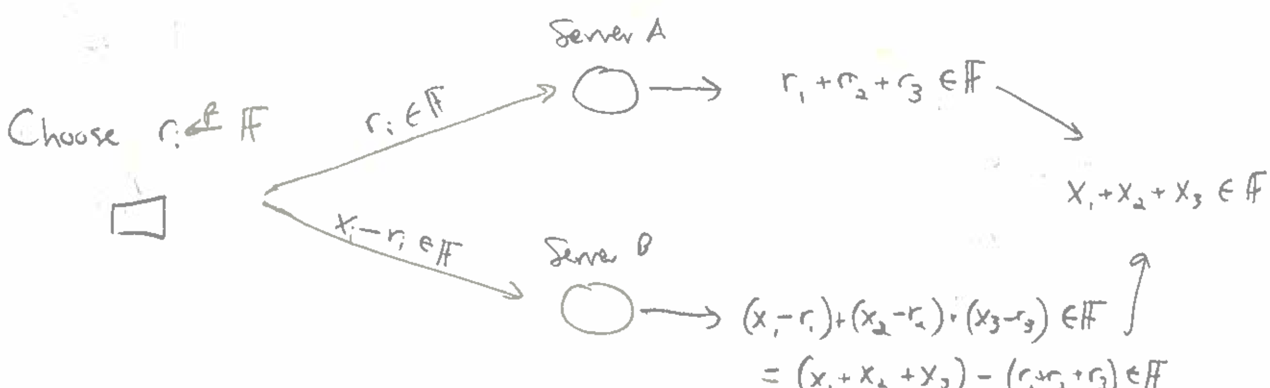


Completeness: Everyone follows protocol $\Rightarrow$ server output $\sum_i x_i$

ZK/Privacy: Each server can simulate view of herself + any # of malicious clients given only $f(x_1, \dots, x_n)$.

Simple Scheme

Very simple protocol achieves this!

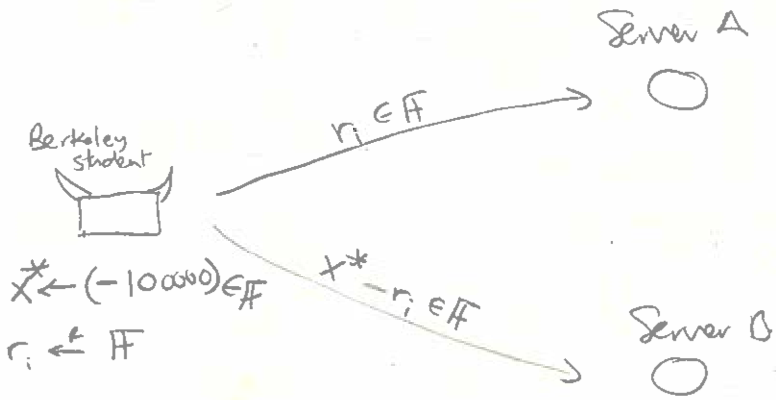


Completeness: ✓

ZK: Each server independently sees all random values, conditioned on sum being $f(x_1, \dots, x_n)$.

Problems w/ Simple Scheme

1) Where do you get 2+ non-colluding servers?
2) Why would Google do this?
3) Evil client



One evil browser can completely screw up the aggregate statistic we wanted to compute!

↳ Can increase it or decrease it by arbitrary amount!

↳ This matters in practice! (private location, private ads, homepages, etc)

We need an extra security property!

Robustness: If the adversary controls m clients ($\{m \cdot n+1, \ldots, n\}$) and the servers execute the protocol correctly, servers output a value in range

$$\left(\sum_{i=1}^{n-m} x_i\right) \le V \le \left(\sum_{i=1}^{n-m} x_i\right) + m$$

Intuition: The worst that evil clients can do is to lie about their value of x_i.

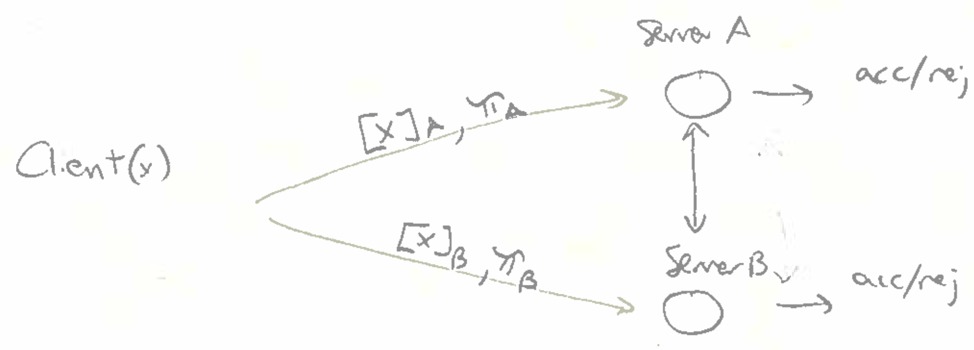
↳ Evil client can always lie about homepage.

The robustness property can be stated more generally for other fns, but let's keep it simple.

How can we get robustness?

↳ Prior approaches used NIZK/SNARKs → Relatively costly (pub-key crypto, etc)

Idea: When client submits secret-shared data to servers, it also submits a .... proof!



In the example here, what is the language $\mathcal{L}$?

$$\mathcal{L} = \{0,1\} \subseteq \mathbb{F} \implies x = 0 \text{ or } x = 1 \text{ in } \mathbb{F}$$

[Evil value $(-100000) \notin \mathcal{L}$.]

Revised Protocol (Prio) - Joint work w/ Dan Boneh

1) Client splits value x into shares $[x]_A, [x]_B$. Sends one share to each server.

2) Client sends proofs π_A, π_B to each server asserting that $[x]_A + [x]_B \in \mathcal{L}$. ⟵ "Valid Submissions"

3) Servers check pf. If it accepts → Keep shares
Else → reject client submission

4) After collecting submissions from all n clients each server publishes sum of shares received so far

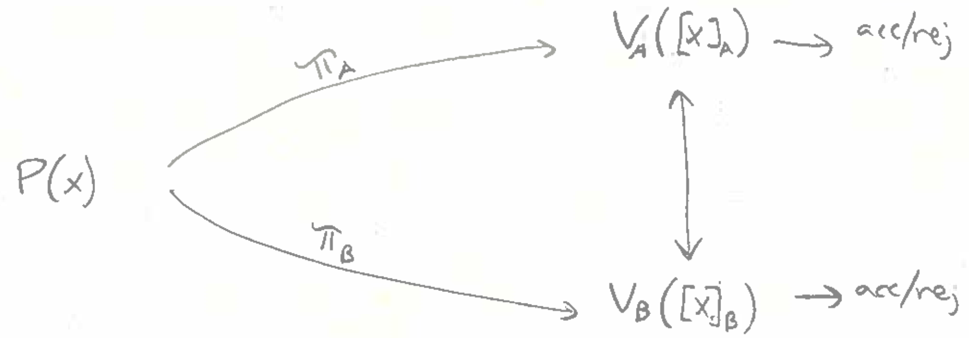
5) Servers recover final sum by combining their sums
$f(x_1, \dots, x_n)$

⇒ Bottom line: Evil clients can't screw up the statistic by "too much"

How do we implement the proof system?

Proofs on Secret-Shared Data

Language $\mathcal{L} \subseteq \mathbb{F}$



Complete If $[x]_A + [x]_B = x \in \mathcal{L} \Rightarrow V_A$ and V_B accept

Sound If $[x]_A + [x]_B = x \notin \mathcal{L} \Rightarrow \forall\, \pi_A^*, \pi_B^*$ V_A and V_B reject

HVZK $\exists$ ppt sim S s.t. for $i \in \{A, B\}$, $\forall\, x \in \mathcal{L}$

$$\{S(i)\} \approx_c \left\{ \mathrm{View}_{V_i}\left[P(x) \leftrightarrow V_A([x]_A) \leftrightarrow V_B([x]_B) \right] \text{ for } \begin{array}{l} [x]_A \xleftarrow{R} \mathbb{F} \\ [x]_B \xleftarrow{R} \mathbb{F} \\ \text{s.t. } [x]_A + [x]_B = x \end{array} \right\}$$

Intuition: Neither server learns anything about x, except that $x \in \mathcal{L}$.

Could consider stronger defn: Full ZK against malicious servers.
↳ Achievable but more complicated.

Fully linear PCPs (BBCGI)

Turns out, very easy to construct from fully linear PCP (GGPR, etc)

For NP language $\mathcal{L}$, recognized by $R(x,w)$, over finite field $\mathbb{F}$

Syntax: $P(x,w) \to \pi$

$V^{x,\pi}(\cdot) \to$ acc/rej. { V makes linear queries to x and π. Outputs $q_i \in \mathbb{F}^{m n}$, expects response $a_i \leftarrow \langle(\pi \| x), q_i\rangle \in \mathbb{F}$ }

FLPCP Properties

Complete: $\forall x \in \mathcal{L}.\ \Pr[V^{x,\pi}(\cdot) = 1 : \pi \leftarrow P(x,w)]$

Sound: $\forall x \notin \mathcal{L}, \forall \pi^*\ \Pr[V^{x,\pi^*}(\cdot) = 1] \leq 1/2$

Strong HVZK: $\exists$ sim S s.t. $\forall\ x \in \mathcal{L}$

$$\{S(\cdot)\} \approx_c \left\{ \begin{array}{l} \text{honest verifier's queries, } q_1, \dots, q_3 \in \mathbb{F}^m \\ \text{query responses } a_1, \dots, a_3 \in \mathbb{F} \text{ for } a_i = \langle(\pi \| x), q_i\rangle \end{array} \right\}$$

Verifier doesn't learn anything about x, apart from fact that $x \in \mathcal{L}$ from looking at query answers.

→ Uses $O(1)$ linear queries, pf size $\Omega(|C|)$ for ckt C, field size $\Omega(|C|)$.

Pfs on shared data are fast!

e.g. ckt with ~6k mul gates

	Prover time (s)		Proof size		S-to-S com
Pi.	0.65	Gap in ver time similar	~1 MB	Huge	~200 bytes
NIZK (dlog)	32				~1 MB
SNARK	~339		288 B		~200 bytes

Ongoing work ☺

Can construct pfs on shared data with unconditional soundness & ZK

↳ No assumptions!

Why is this surprising?

Common misconception was that NIZK/SNARK/full MPC was necessary

Constructing Proof on Shared Data From FLPCP

$P(x,w)$ $\xrightarrow{\pi_A}$ $V_A([x]_A)$

$P(x,w)$ $\xrightarrow{\pi_B}$ $V_B([x]_B)$

$\pi \leftarrow P_{FLPCP}(x,w) \in \mathbb{F}^m$

Choose random $\pi_A, \pi_B \in \mathbb{F}^m$ s.t.

$\pi = \pi_A + \pi_B$

$V_A([x]_A) \xrightarrow{[a_1]_A, \dots [a_3]_A}$ Run $V_{FLPCP}^{x,\pi}$

$V_B([x]_B) \xrightarrow{[a_1]_B, \dots [a_3]_B}$ Run $V_{FLPCP}^{x,\pi}$

To check the proof the verifiers V_A, V_B run the FLPCP verifier on shared randomness

$V_{FLPCP} \xrightarrow{q_i \in \mathbb{F}^{m+n}}$

$\xleftarrow{a_i = \langle (\pi | x), q_i \rangle \in \mathbb{F}} V_{FLPCP}$

V_A and V_B need to be able to respond to the FLPCP verifier's queries.

Remember: Computing linear functions on additively shared data is easy!

↳ This goes back to the lectures on MPC

Each verifier holds:
* a share of $x \in \mathbb{F}$
* a share of $\pi \in \mathbb{F}^m$
* each query vector $q_i \in \mathbb{F}^{m+1}$

↳ Each verifier can <u>locally</u> <u>compute</u> share of answer

$$a_i = \langle (\pi | x), q_i \rangle \in \mathbb{F} \ !$$

Verifiers broadcast their shares of query answers, feed them to V_{FLPCP}, output whatever it outputs

→ Completeness, soundness, HVZK follow easily

So now we have compiler

FLPCP → Proof on secret-shared data → Private agg. w/ robustness

Where do we get a fully linear PCP?

The standard LPCP constructions that David presented last week are also fully linear!

Putting it all together:

IF validity predicate is computed by ckt C

Client-Server Comm	Server-to-Server Com	Field size	Client/Server Compute
$O(\lvert C\rvert)$	$O(1)$	$\Omega(\lvert C\rvert)$	$O(\lvert C\rvert)$

Very small # of bits communicated b/w the aggregation servers! Indep of $\lvert C\rvert$, almost

No public-key crypto or assumptions needed! This is much faster than SNARKs for client/prover
↳ Downside is that C-to-S comm. is larger (ongoing work)

What if you want a more complicated agg statistic?

- Lin regression
- STDDEV
- Mode
- Most popular heavy hitters

↳ Use "linear" data structures... reduce problem of computing f(·) to problem of computing sums