Today
* Recap: SNARKs & Linear ACs
* Private aggregation
* Simple scheme & its problems
* Fix: proofs on secret-shared data
* Fully linear ACs

(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 class

Today, we are covering results that are "hot off the press:"
- Application of very recent techniques to privacy problem
  Browser vendor compacting the most popular homepage w/o revealing 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: Graph 3-coloring

Normal NP proof: \[ P \text{ 3-colors } G, R(M) \text{ bits } \rightarrow V \]

SNARG

\[ P \text{ snarg of } G, 8A \text{ bits } \rightarrow V \]

\[ \text{No matter how large graph is!} \]

- Proof shorter than NP witness!
- Good evidence that SNARGs don't exist for all NP lang under "standard" assumptions

\[ \text{What is this?} \]

We constructed SNARG using general strategy (BCIOp,...)

\[ \text{Linear PCP for } L \]

\[ \text{Crypto compiler using linear-only encryption} \]

\[ \text{SNARG for } L \]

\[ \text{Info-theoretic part (no assumptions)} \]

\[ \text{Crypto part (uses assumptions)} \]

Since we will be using Linear PCPs again today, want to refresh your memory.
Types of Proof (into a language $L \subseteq \{0,1\}^*$) 

<table>
<thead>
<tr>
<th>Type</th>
<th>Access to $x$</th>
<th>Access to $\Pi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP/MA</td>
<td>Read all</td>
<td>Read all</td>
</tr>
<tr>
<td>PCP</td>
<td>Read all</td>
<td>Point query</td>
</tr>
<tr>
<td>Linear PCP</td>
<td>Read all</td>
<td>Linear query</td>
</tr>
<tr>
<td>Fully linear PCP</td>
<td>Linear query</td>
<td>Linear query</td>
</tr>
<tr>
<td>PCPP</td>
<td>Point query</td>
<td>Point query</td>
</tr>
</tbody>
</table>

Many more! Also interactive!

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

Q: Can you construct a FL-PCP for $L$ from an NP proof for $L$?
Private Aggregation

Let $\mathcal{F} : \mathbb{F}^n \to \mathbb{F}$ 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 could this be useful?

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

$F()$ computes average speed

$\Rightarrow$ 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.

$F()$ computes sum of $x_i$

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

E.g. $x_i$ is location of phone i

$F()$ computes most popular value amongst inputs

$\Rightarrow$ Learn pop location w/o leaking any individual location.

Two general approaches

1. Local differential privacy
2. MPC-based}
We'll simplify the problem a bit

1) We'll use $2^n$ "non-colluding" servers for practical reasons...
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

Server want to compute $f(x_1, ..., x_n) = \sum_{i=1}^{n} x_i \in \mathbb{F}$ (Popularity of

**Completeness:** Everyone follows protocol $\Rightarrow$ server output $\sum x_i$.

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

**Simple Scheme**

Very simple protocol achieves this!

Choose $r_1 \in \mathbb{F}$

Choose $r_1 \in \mathbb{F}$

\[\text{Server A} \quad r_1 + r_2 + r_3 \in \mathbb{F} \]

\[x_i, x_2, x_3 \in \mathbb{F} \]

\[\text{Server B} \quad (x_i - r_1 + (x_2 - r_2) + (x_3 - r_3) \in \mathbb{F} \]

\[= (x_i + x_2 + x_3) - (r_1, r_2, r_3) \in \mathbb{F} \]

**Completeness:**

\[\checkmark \]

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

17
Problems w/ Simple Scheme

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

\[
\begin{align*}
\text{Berkeley student} & \quad \xrightarrow{\mathbf{r} \in \mathbb{F}} \quad \text{Server A} \\
x^* \leftarrow (-100000) \in \mathbb{F} & \quad x^* \xrightarrow{\mathbf{r} \in \mathbb{F}} \quad \text{Server B} \\
\mathbf{r} \leftarrow \mathbb{F} & \quad \text{Server A}
\end{align*}
\]

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, homepage, etc)

We need an extra security property!

Robustness: If the adversary controls m clients' and the servers execute the protocol correctly, servers output a value in range

\[
\left(\sum_{i=1}^{m} x_i\right) \leq V \leq \left(\sum_{i=1}^{m} x_i\right) + m
\]

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 cases, but let's keep it simple.

How can we get robustness?
- Prior approaches used NIZK/SNARKs \(\Rightarrow\) relatively costly (pub-key crypto, etc.)
Idea: When client submits secret-shared data to server, it also submits a proof of prover A.

\[ \text{Client(s)} \quad \xrightarrow{\ [x]_A, \tau_A} \quad \text{Server A} \quad \xrightarrow{\text{acc/reg}} \]

\[ \xrightarrow{\ [x]_B, \tau_B} \quad \text{Server B} \quad \xrightarrow{\text{acc/reg}} \]

In the example here, what is the language \( L \)?

\[ L = \{0, 1\} \subseteq \mathbb{F}, \quad \Rightarrow \quad x = 0 \text{ or } x = 1 \text{ in } \mathbb{F} \]

[Evil value (-10000) \not\in L]

Revised Protocol (P1is) - 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 \( \tau_A, \tau_B \) to each server assuring that \([x]_A + [x]_B \in L \). \( \leftarrow \) "Valid Submissions"

3. Servers check proofs. 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. Server recovers sum by combining their sum
   \[ f(x_1, \ldots, x_n) \]

\( \Rightarrow \) 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 $L \subseteq \mathbb{F}$

$P(x) \xrightarrow{\Pi_A} V_A([x]_A) \rightarrow \text{acc/rej}$

$\xrightarrow{\Pi_B} V_B([x]_B) \rightarrow \text{acc/rej}$

**Correct** IF $[x]_A + [x]_B = x \in L \Rightarrow V_A$ and $V_B$ accept

**Sound** IF $[x]_A + [x]_B = x \notin L \Rightarrow \forall \Pi_A, \Pi_B^*, V_A$ and $V_B$ reject

$\Pi_V \kappa \equiv \text{rt sim } S \text{ s.t. } S_0 \leftarrow \{ \Pi_A, \Pi_B \}, \forall x \in L$

$\{ S_0(x) \} \Leftarrow \{ \text{V: ev: } (P(x) \xrightarrow{\Pi_A} V_A([x]_A)) \}

\{ \text{S: ev: } (V_B([x]_B)) \}

s.t. $[x]_A \in \mathbb{F}$

$s.t. [x]_B \in \mathbb{F}$

$s.t. [x]_A + [x]_B = x$

Intuition: neither server learns anything about $x$, except that $x \in L$.

Could consider stronger defin: Full ZK against malicious servers.

Achievable but more complicated.
Turns out, very easy to construct from Sully linear PCP. For NP language \( L \), recognized by \( R(x,w) \), over finite field \( \mathbb{F} \):

- Syntax: \( P(x,w) \rightarrow \hat{y} \)
  - \( V^x,\hat{y}() \rightarrow \text{acc/rej.} \)
  - \( \mathbb{V} \), makes linear queries to \( x \) and \( \hat{y} \).

- \( \mathbb{F} \)PCP Property:
  - Complete: \( \forall x \in L. \quad P[x,y] = 1 \quad \Rightarrow \hat{y} = R(x,w) \)
  - Sound: \( \forall x \notin L, \forall y \quad \Pr[V^x,y()(...) = 1] \leq \frac{1}{2} \)

- Strong \( \text{HVZ1k} \): \( \exists \text{ sim.} \quad \forall x \in L \)
  - \( \{S(x)\} \) is \( \{ \text{honest verified queries, } q_1, \ldots, q_i \in \mathbb{F}^n \} \)
  - \( \{ \text{query responses, } a_1, \ldots, a_i \in \mathbb{F} \} \quad \text{for } o_i = \langle \mathbb{K}, q_i \rangle \)

Verifier doesn't learn anything about \( x \), apart from \( \mathcal{S}(x) \), from looking at query answers.

- Uses \( O(1) \) linear queries, size \( \mathcal{S}(x) \) s.t. \( \hat{y} \) is \( \mathcal{F} \)pcp.
- \( \hat{y} \)s on shared data are fast:
  - e.g., \( \hat{y} \) with \( n \)Gt mul gates
  - \( \mathcal{P}(n) \) s.t.
  - NICE \( (\log n) \) : 32
  - SNARK \( n \) : 339

- Proof size \( \approx 2 \) MB
- \( \hat{y} \) size \( \approx 1 \) MB
- \( S \)-to-\( S \) comm.
  - \( \approx 200 \) bytes
  - \( \approx 200 \) bytes

Ongoing work

Can construct \( \hat{y} \)s on shared data with unconditional soundness \& \( \mathcal{F} \)pcp

\( \Rightarrow \) No assumptions.

Why is this surprising?

Common misconception was that NICE/SNARK/SNAIL MTC

... necessary.
Construcing Proof on Shared Data From FLPCP

\[
P(x,w) \xrightarrow{\pi} V_A([x]_A) \quad \downarrow \quad [a_1]_A, \ldots, [a_3]_A
\]

Choose random \( \pi, \pi_A, \pi_B \in \mathbb{F}_t^m \)
\( \pi = \pi_A \times \pi_B \)

Run \( V_{FLPCP} \)
\[
[\pi]_B, \ldots, [\pi]_B
\]

\( V_B([x]_B) \)

To check the proof the verifiers \( V_A, V_B \) run the FLPCP verifier on shared randomness.

\[
q_i \in \mathbb{F}_t^{m+n}
\]

\( V_{FLPCP} \xrightarrow{a_i = <(\pi, q), q_i> \in \mathbb{F}} \)

\( V_A \) and \( V_B \) need to be able to respond to the FLPCP verifiers' 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 \( \pi \in \mathbb{F}_t^m \)
* Each query vector \( q_i \in \mathbb{F}_t^{m+n} \)

Each verifier can locally compute share of answer
\[
a_i = <(\pi, q), q_i> \in \mathbb{F}
\]

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

\( \rightarrow \) Completeness, soundness, HWE follow easily
So now we have compiler

Where do we get a fully linear PCP?

The standard 2PCP constructions that David presented last week are also fully linear!

Putting it all together:

If validity predicate is computed by circuit $C$

<table>
<thead>
<tr>
<th>Client-Serve</th>
<th>Serve $\rightarrow$ Serve</th>
<th>Field size</th>
<th>Client-Serve</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1Ci)$</td>
<td>$O(Ci)$</td>
<td>$\Omega(1Ci)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Very small # of bits communicated b/w the aggregation servers! Indeed of $1Ci$, almost.

No public-key crypto or assumptions needed! This is much faster than $\sum_{i,j}$. Conclude for client/privacy.

$\Rightarrow$ Downside is that $Cu-S$ comm. is larger (ongoing work)

What if you want a more complicated agg statistic?

- Lin. regression
- $\text{STDEV}$
- Mode
- Most popular/long hitters

Use "linear" data structures... reduce problem of computing $f(1)$ to problem of computing $\sum$'s.