Lecture 10: Three-party computation

MIT - 6.893

Fall 2020

Henry Corrigan-Gibbs

Plan

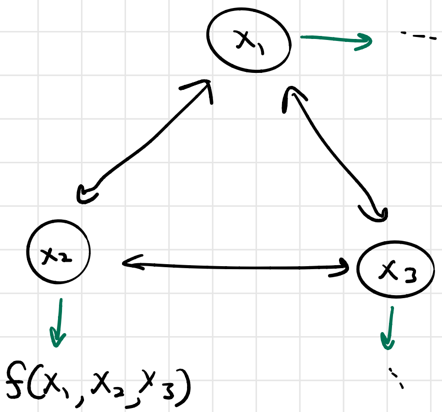
* Recap: MPC Background
* Arithmetic circuits
* 3PC protocol
* 3PC security analysis

Logistics

* HW3 due Friday 5pm via Gradescope
* Guest lecture Susan McGregor tomorrow — PLEASE DO READINGS.
* OH tomorrow 3pm-4:30pm

Recap: MPC

* n Parties, each with private input: $x_1, \ldots, x_n$
* Want to compute a public fn $f(x_1, \ldots, x_n)$ of their private data.



Parties "learn nothing more" about each other's inputs than $f(x_1, \ldots, x_n)$.

→ Many types of MPC!

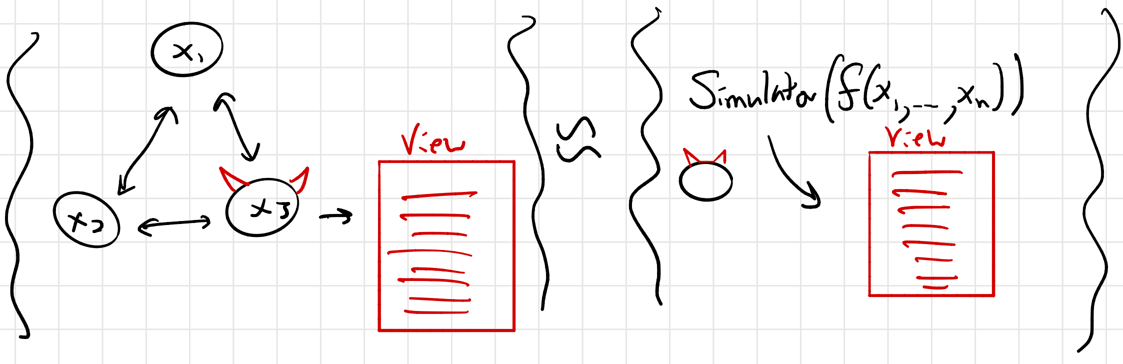
Today: 3PC with semi-honest security with honest majority (≤ 1 corrupt party) & inf. theoretic security.

Recap: Simulation

Way to capture the notion that adv
"Learns nothing except $f(x_1, \ldots, x_n)$".

If adv can simulate its view of interaction without $(x_1, \ldots, x_n)$ — only given $f(x_1, \ldots, x_n)$ — then adv cannot (intuitively) have learned anything about $(x_1, \ldots, x_n)$, apart from what $f(x_1, \ldots, x_n)$ leaks.

Real World Ideal World



Security Definition

An MPC protocol π securely realizes f with semi-honest security if $\exists$ sim S s.t.

$\forall$ subsets $C \subseteq [n]$ w/ $|C| < n/2$ and
$\forall$ inputs $(x_1, \ldots, x_n)$

$$\underbrace{\left\{ \begin{matrix} \text{Views of parties in } C \\ \\ \text{(outputs of all parties)} \end{matrix} \right\}}_{\text{"Real"}} \simeq \underbrace{\left\{ \begin{matrix} \mathrm{Sim}\left(\begin{matrix} C, \{x_i \mid i \in C\} \\ f(x_1, \ldots, x_n) \end{matrix} \right) \\ \\ f(x_1, \ldots, x_n) \end{matrix} \right\}}_{\text{"Ideal"}} .$$

Addition to def'n from last time: by sticking outputs in here, we guarantee that parties get right output (correctness).

We will see a 3PC protocol...

* semi-honest secure
* requires honest majority (static adv)
* info theoretic security.

(Relevant work: BGW, CCD, Beaver, ...)

For this protocol (& many MPC protocols), represent fn f as a circuit.

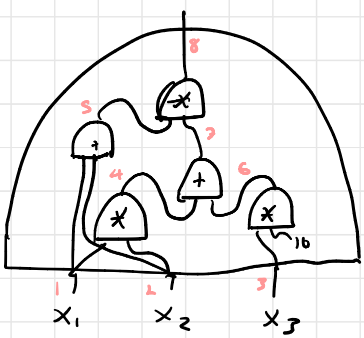
Arithmetic ckt over finite field $\mathbb{F}$. ← Will identify w/ ints mod p

↳ Think: + and * modulo prime p.

$a + b \in \mathbb{F} \simeq a+b \bmod p$
$a * b \in \mathbb{F} \simeq a * b \bmod p$

Arithmetic ckt over $\mathbb{F}$: Circuit where gates are +, *, and scalar-multiplications.
↳ wires carry values in $\mathbb{F}$ (ie. ints mod p).

$$f(x_1, x_2, x_3) := (x_1 \cdot x_2 + 10x_3) \cdot (x_1 + x_2)$$



NOTE: All arithmetic here is in $\mathbb{F}$ (modulo p, if you prefer)

Useful life fact

If language $L \in P$ (poly time) then there is a poly-sized logspace-uniform ckt C_L s.t.

$$x \in L \iff C_L(x) = 0 .$$

See Arora and Barak Thm 6.7

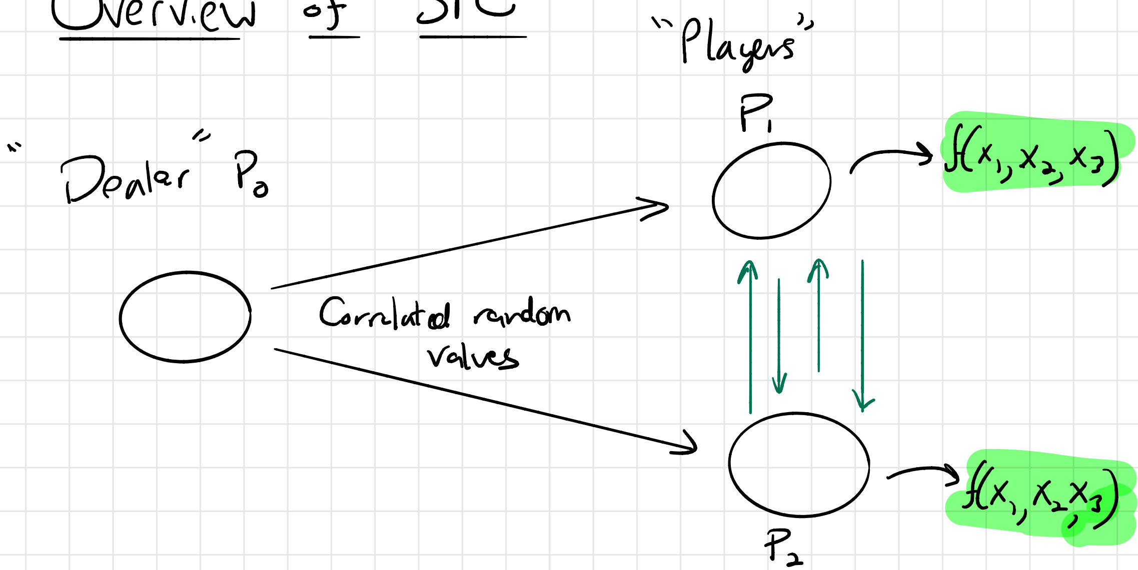
Boolean ckts are arithmetic ckts over $\mathbb{F}_2$.
Similar result holds over larger fields.

$\Rightarrow$ So, if we want to compute any poly-time fn on data $(x_1, \ldots, x_n)$ in MPC, we can do so w/ an arithmetic ckt computation.

Can label wires in ckt from inputs to outputs in topological order.

↳ Labeling is common to all players.

Overview of 3PC



1. Dealer sends some randomness to P_1, P_2
2. Players P_1 and P_2 run computation.

Complexity

- Communication $\propto$ size of ckt computing f
- # of comm rounds $\propto$ dept of ckt computing f.

"Gate-by-Gate" Strategy

INPUT

- Players start out holding additive shares of values on all input wires.

COMPUTATION

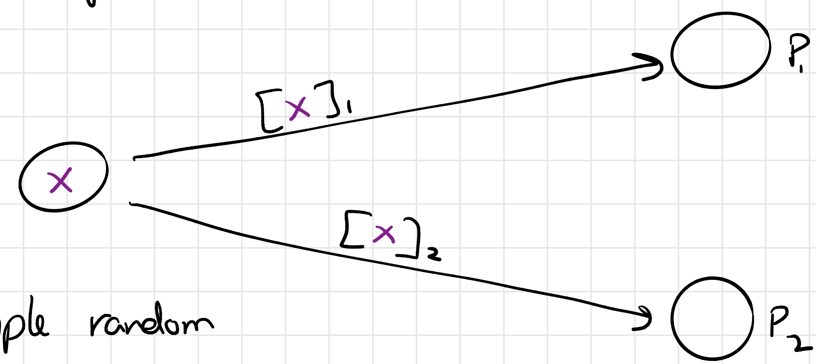
- Players walk through each wire of ckt in topological order, computing shares of that wire's value.

OUTPUT

- Once players have shares of output wire value, can publish it to learn $f(x_1, \ldots, x_n)$.

Input Phase:

All parties need shares of all other parties' inputs



Sample random

$$[x]_1, [x]_2 \xleftarrow{R} \mathbb{F}$$

s.t. $[x]_1 + [x]_2 = x$

Other parties do the same...

Computation Phase

Only need to handle 3 gate types

1. Add
2. Mul by scalar
3. Mul

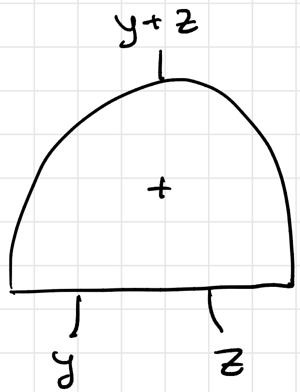
ADD Gate

$[y]_1, [z]_1 \mapsto [y]_1 + [z]_1 = [y+z]_1$

P_1

$[y]_2\ [z]_2 \mapsto [y]_2 + [z]_2 = [y+z]_2$

P_2



Notice that we get sharing of $y+z$ since

$$[y+z]_1 + [y+z]_2 = [y]_1 + [z_1] + [y]_2 + [z]_2$$
$$= y+z$$

Can add shares of zero to rerandomize.

MUL by Scalar

Just multiply by constant $c \in \mathbb{F}$ locally

$$[y]_1 + [y]_2 = y$$

$$c[y]_1 + c[y]_2 = cy$$

$$\Longrightarrow c[y]_i = [c \cdot y]_i$$

MUL: What doesn't work ...

For addition, parties added shares locally.

For multiplication, multiply locally? Problem!

$$[y]_1 \cdot [z]_1 + [y]_2 \cdot [z]_2 \neq y \cdot z$$

Need $y \cdot z = ([y]_1 + [y]_2) \cdot ([z]_1 + [z]_2)$

$$= [y]_1 \cdot [z]_1 + \underline{[y]_1 [z]_2} + \underline{[y]_2 \cdot [z]_1} + [y]_2 \cdot [z]_2$$

MUL

So far, the players haven't needed to communicate.

For multiplications, they do.

For each mul gate, dealer sends to P_1, P_2 additive shares of values $a, b, c \in \mathbb{F}$ s.t.

$$a \cdot b = c \in \mathbb{F}.$$

So, P_1 has $[a]_1, [b]_1, [c]_1$ s.t. $([a]_1 + [a_2])([b]_1 + [b]_2) = [c]_1 + [c]_2 \in \mathbb{F}$

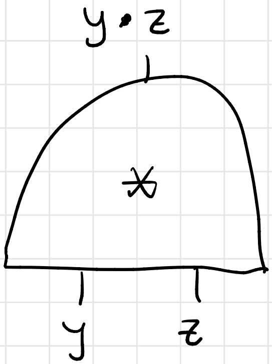
P_2 has $[a]_2\ [b]_2\ [c]_2$

Known as "Multiplication triples" or "Beaver triples"

MUL (cont'd)

Players start out holding shares of y, z.

They want shares of $y \cdot z$.



Steps:

1. For each $i \in \{1,2\}$, P_i publishes

$$[d]_i \leftarrow [y]_i - [a]_i$$

$$[e]_i \leftarrow [z]_i - [b]_i$$

2. Players reconstruct

$$d \leftarrow [d]_1 + [d]_2$$

$$e \leftarrow [e]_1 + [e]_2$$

3. Players compute shares of yz as

$$[yz]_i \leftarrow de/2 + d[b]_i + e[a]_i + [c]_i$$

$$[yz]_1 = de/2 + d[b]_1 + e[a]_1 + [c]_1$$

$$[yz]_2 = de/2 + d[b]_2 + e[a]_2 + [c]_2$$

$$= de + (y-a)b + (z-b)a + ab$$

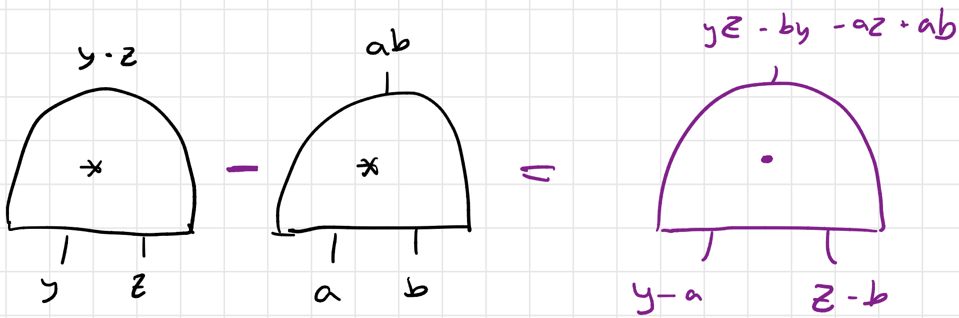
$$= (y-a)(z-b) + (y-a)b + (z-b)a + ab$$

$$= yz - \cancel{az} - \cancel{by} + \cancel{ab} + \cancel{yb} - \cancel{ab} + \cancel{az} - \cancel{ab} + \cancel{ab}$$

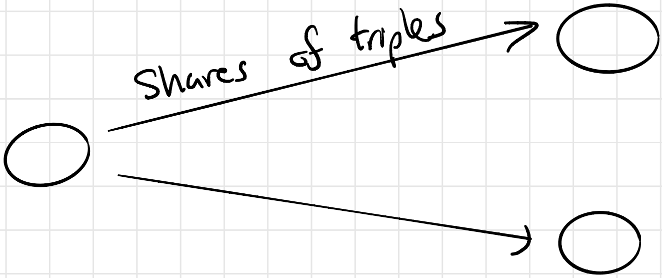
$$= yz$$

!

Where did that come from?



Summary



1. Dealer sends to players shares of values (a, b, c) ... one per gate in ckt.

2. All parties send shares of their inputs to P_1, P_2.

3. Players P_1 and P_2 walk through ckt gate by gate, computing shares of internal wire values.

   Add, mul by scalar → no comm

   Mul → one round of comm

4. Finally, players broadcast output shares.

Security

Need to construct a simulator that outputs view of each of 3 parties.

Dealer → Direct to simulate

Player → Output random values in $\mathbb{F}$ for all field elements up to last set of shares which sum to $f(x_1, \ldots x_3)$.

To argue simulation is correct, notice that all values broadcast are blinded by random values (used only once)

Making these arguments formal is tricky.

In malicious model, it's very subtle.

Notes:

- Dealer does almost nothing.
  - ↳ Can replace dealer w/ crypto assumptions.
- Very cheap in computation... provided that your computation has a "nice" representation as a small ckt.
- Not malicious secure. Why?
- Triples-based approach generalizes to any # of parties.