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 function $S(x_1, \ldots, x_n)$ of their private data.

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

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.
Security Definition

An MPC protocol \( \Pi \) securely realizes \( f \) with semi-honest security if \( \exists \) sim \( S \) s.t.

\[
\forall \text{ subsets } C \subseteq [n] \quad \forall \left| C \right| \leq n/2 \quad \text{and} \quad \forall \text{ inputs } (x_1, \ldots, x_n)
\]

\[ \text{“Real”} \quad \left\{ \text{Views of parties in } C \right\} = \left\{ \text{outputs of all parties} \right\} \]

\[ \text{“Ideal”} \quad \sim \left( C, \{x_i : i \in C^c\} \right) \]

\[ \sim \left( f(x_1, \ldots, x_n) \right) \]

Addition to defn from last line: by sticking outputs in here, we guarantee that parties get right output (correctness).
We will see a BPC protocol...

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

(Relevant work: BGU, CC, Beaver, ...)

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

Arithmetic ckt over finite field $\mathbb{F}$:

Think: $+$ and $\times$ modulo prime $p$.

\[
\begin{align*}
a + b \in \mathbb{F} & \Rightarrow a + b \mod p \\
a \times b \in \mathbb{F} & \Rightarrow a \times b \mod p
\end{align*}
\]

Arithmetic ckt over $\mathbb{F}$: circuit where gates are $+$, $\times$, and scalar multiplications.

Wires carry values in $\mathbb{F}$ (i.e., ints $\mod p$).

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

\[\text{Note: All arithmetic here is in } \mathbb{F} \text{ (modulo } p \text{, if you prefer)}\]
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 ckt's are arithmetic ckt's over $\mathbb{F}_2$. Similar result holds over larger fields.

$\Rightarrow$ So, if we want to compute any poly-time fn on data $(x_1, \ldots, x_m)$ 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

"Dealer" \( P_0 \)

Correlated random values

Players

1. Dealer sends some randomness to \( P_1, P_2 \)

2. Players \( P_1 \) and \( P_2 \) run computation.

Complexity

- Communication \( \propto \) size of circuit computing \( f \)
- \# of comm rounds \( \propto \) depth of circuit computing \( f \).
"Gate-by-Gate" Strategy

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

**Computation**
- Players walk through each wire of CBT in topological order, computing shares of that wire’s value.

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

---

**Input Phase:**
All parties need shares of all other parties’ inputs

Sample random

\[
[x]_1, [x]_2 \leftarrow \text{TF}
\]

\[s.t. \quad [x]_1 \cdot [x]_2 = x \]

---

Other parties do the same...
Computation Phase

Only need to handle 3 gate types

1. Add
2. Mul by scalar
3. Mul

ADD Gate

\[ \begin{align*}
[y], \quad [z], \\
\circ P_i \\
[y], \quad [z]_2 \\
\circ P_j
\end{align*} \]

\[ \rightarrow \begin{align*}
[y], \quad [z] = [y + z], \\
[y], \quad [z] = [y + z]
\end{align*} \]

Notice that we get sharing of \( y + z \) since

\[ \begin{align*}
[y + z], \quad [y + z]_2 \\
= \begin{align*}
[y], \quad [z]_1 \cdot [z]_2 \\
= y + z
\end{align*}
\]

Can add shares of zero to re-randomize.
MUL by Scalar

Just multiply by constant c ∈ F locally

\[ [y]_1 + [y]_2 = y \]

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

\[ \Rightarrow c[y]_i = [c \cdot y]_i \]

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

\[ = [y]_1 \cdot [z]_1 + [y]_1 [z]_2 + [y]_2 [z]_1 + [y]_2 [z]_2 \]
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. $(a_1 + [a_2])(b_1 + [b_2]) = [c_1] + [c_2] \in \mathbb{F}$

Known as "Multiplication triples" or "Beaver triples"
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
   
   $[yz]_i \leftarrow de/2 + d[b]_i + e[a]_i + [c]_i$
\[
[yz]_1 = de + d[b] + e[a] + [c]
\]
\[
[yz]_2 = de + d[b] + e[a] + [c]
\]

\[
= de + (y-a)b + (z-b)a + ab
\]
\[
= (y-a)(z-b) + (y-a)b + (z-b)a + ab
\]
\[
= y^2 - ax - by + ab + yb - z^2 + z^2 - az + ab + ab
\]
\[
= y^2
\]

*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 valves.

   - \(\text{Add, mul by scalar } \rightarrow \text{no comm}\)
   - \(\text{Mul } \rightarrow \text{one round of comm}\)

4. Finally, players broadcast output shares.
Need to construct a simulator that outputs view of each of 3 parties.

Dealer → Direct to simulate

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

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.