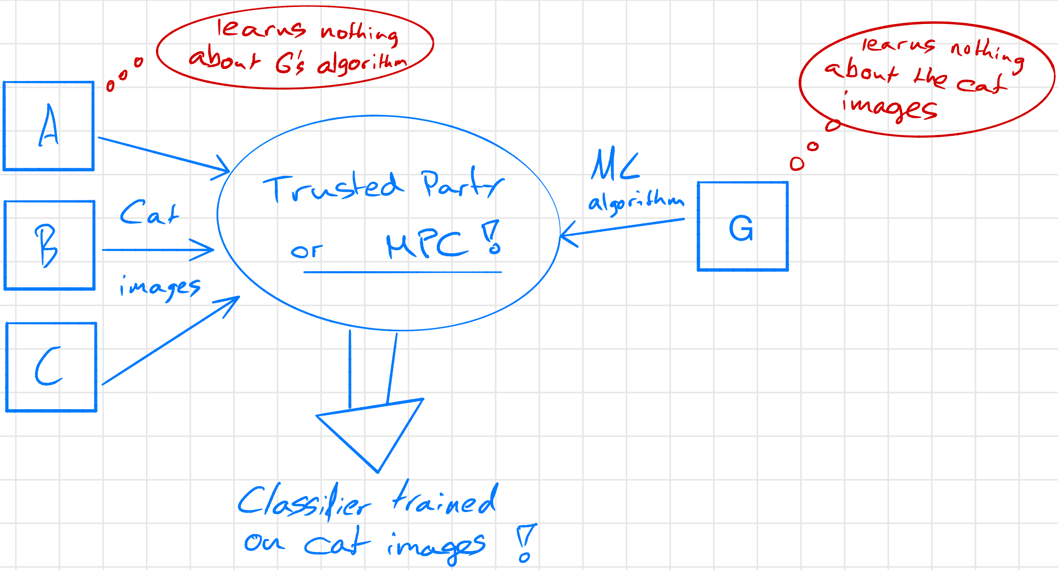
CS 355 Lecture 7: Multiparty Computation

Previously:

- Interactive protocols for proofs:
  - → What about more general protocols?
  - → what about $n > 2$ parties?
- Secret Sharing:
  - → our first n-party functionality!
  - → can we do more than merely share secrets?

Today: Multiparty Computation (MPC)

Anything that can be computed with a trusted third party can also be securely computed without!



Other Applications :

* E-voting
* Private auctions
* etc.

Q: Why don't we use MPC for EVERYTHING ?

A: It's not very efficient (~ 100-1000x overhead for ML)

Defining MPC

There are n parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$ that want to jointly compute a function

$$y = f(x_1, \ldots, x_n)$$

we can generalize this so each party gets its own output y_i

The adversary corrupts a subset of the parties and makes them collude to break security of the protocol.

There are two main security models for MPC:

Semi-honest: The corrupted parties follow the protocol specification exactly. After the protocol completes, they look at the transcript and try to extract information about the honest parties' inputs.

Malicious: The corrupted parties may arbitrarily deviate from the protocol specification at any time, to learn extra information about the honest parties' inputs or fool them into producing the wrong output.

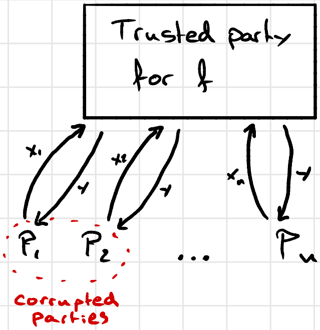
The verifier in an HVZK proof is an example of a semi-honest adversary.

For this lecture, we'll focus on the semi-honest setting.

Defining security for semi-honest MPC:

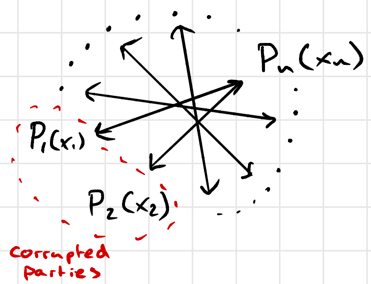
Informally: "anything the adversary learns in an execution of the MPC protocol, it could also have learned if all parties were interacting with a trusted third party"

ideal world



$\approx$

real world



This is also called the "real-ideal paradigm".

What does the adversary learn in the ideal world?

* The inputs of corrupted parties
* The output of the computation

we'll get back to this in next week's lecture on differential privacy

Formally, if C is the set of corrupt parties, there exists an efficient simulator Sim such that for all functions f and inputs $x_1, \dots, x_n$:

$$\mathrm{Sim}\Big(C, \underbrace{\{x_i : i \in C\}}_{\text{the inputs of corrupt parties}}, \underbrace{y = f(x_1, \dots, x_n)}_{\text{the output of the computation}}\Big) \approx_c \underbrace{\{V_i : i \in C\}}_{\text{the view of } A \text{ in a real execution of the protocol}}$$

Recap: additive secret sharing

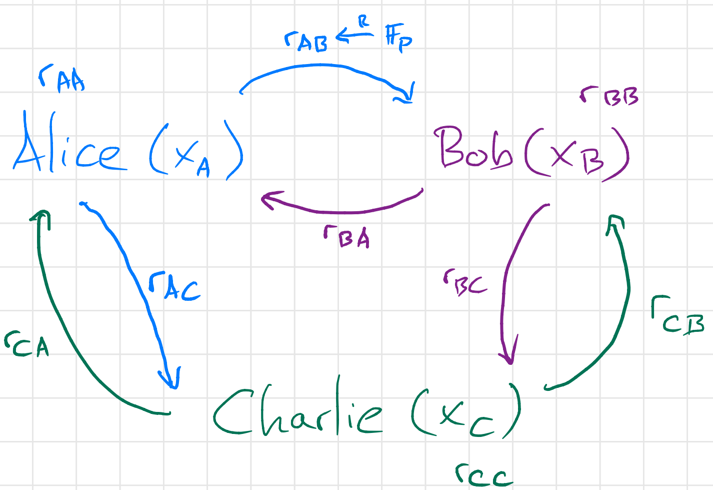
To share $s \in \mathbb{F}_p$ (a field of prime order e.g. $\mathbb{Z}_p$ for p prime) among n parties, sample $r_1, \ldots, r_{n-1} \xleftarrow{R} \mathbb{F}_p$ and set $r_n = s - \sum_{i=1}^{n-1} r_i \in \mathbb{F}_p$

We use $[s]$ to denote additive secret sharing of s

$$[s] = (r_1, r_2, \ldots, r_n)$$

MPC by computing on secret-shared data

- Each party has an input $x_i \in \mathbb{F}_p$
- The function f is represented as an arithmetic circuit over $\mathbb{F}_p$ (i.e. a circuit with addition and multiplication gates over $\mathbb{F}_p$).
- The parties start by secret sharing their inputs:



Alice: Chooses $r_{AB}, r_{AC} \xleftarrow{R} \mathbb{F}_p$ and sends one share to Bob and one to Charlie.

Alice's share is $r_{AA} = x_A - r_{AB} - r_{AC}$

If we can perform additions and multiplications over secret-shared data, we get a MPC protocol secure against semi-honest adversaries:

1. Each party secret shares their input with every other party
2. For each addition gate in the circuit, with inputs $[a]$, $[b]$ the parties compute shares of $[a+b]$

   the inputs of the gate are secret shared

   the parties have a secret sharing of the output of the gate
3. For each multiplication gate, with inputs $[a]$, $[b]$ the parties compute shares of $[ab]$
4. Each party publishes their share of the output value so each party can reconstruct the output.

<u>Example:</u> $f(x_A, x_B, x_C) = (x_A + x_B) \cdot x_C$

Alice:	r_{AA}, r_{BA}, r_{CA}	addition gate $\Longrightarrow$	r'_A, r_{CA}	multiplication gate $\Longrightarrow$	r''_A
Bob:	r_{AB}, r_{BB}, r_{CB}		r'_B, r_{CB}		r''_B
Charlie:	r_{AC}, r_{BC}, r_{CC}		r'_C, r_{CC}		r''_C
	$[x_A]$ $[x_B]$ $[x_C]$		$[x_A+x_B]$ $[x_C]$		$[(x_A+x_B)\cdot x_C]$
	the parties start by secret sharing their inputs (i.e., $x_A = r_{AA} + r_{AB} + r_{AC}$)		by some means, the parties end up with shares of $x_A + x_B$ (i.e., $x_A + x_B = r'_A + r'_B + r'_C$)		by some means, the parties end up with shares of the output $y = (x_A+x_B)\cdot x_C$ (i.e., $y = r''_A + r''_B + r''_C$)

Additions of shares

Given shares of a and b, $[a+b] = [a]+[b]$ component-wise addition

If $[a] = \underbrace{(a_1, \ldots, a_n)}_{\text{additive shares of } a}$ where $\sum_{i=1}^n a_i = a \in \mathbb{F}_p$
$[b] = (b_1, \ldots, b_n)$ where $\sum_{i=1}^n b_i = b \in \mathbb{F}_p$

Then $[a+b] = (a_1+b_1, \ldots, a_n+b_n)$ satisfies $\sum_{i=1}^n (a_i+b_i) = a+b \in \mathbb{F}_p$

Similarly:
- scalar multiplication: $[ka] = k \cdot [a]$
- addition by constant: $[a + k/n] = k + [a]$

alternatively, one party can add k to their share and all other parties do nothing

How do we multiply shares?

- Using public-key crypto
  - * Oblivious Transfer
  - * Somewhat Homomorphic Encryption

  (this gives computationally secure MPC for a semi-honest adversary that corrupts $n-1$ parties)

- Using Shamir Secret Sharing

  (this gives information theoretically secure MPC for a semi-honest adversary that corrupts $< n/2$ parties)

Information theoretic MPC

(or, how to multiply additive secret shares using Shamir secret sharing)

Recap: t-out-of-n Shamir Secret Sharing

secret $\in \mathbb{F}_p$

random poly of degree $< t$

$$f(x) = s + x + c_1 x^2 + \dots + c_{t-1} \cdot x^{t-1}$$

shares: $y_i = f(i)$ for $i \in [n]$

reconstruction: for any subset of t parties (e.g. $1, \dots, t$):

Vandermonde matrix

$$V^{-1} \circ \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix} = \begin{bmatrix} s \\ c_1 \\ \vdots \\ c_{t-1} \end{bmatrix}$$

In particular,

this is also a t-out-of-t additive sharing of s!

$$s = (V^{-1})_1 \circ \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix} = \overbrace{\sum_{i=1}^{t} \lambda_i \cdot y_i}$$

these are the elements in the first row of V^{-1}. They are also called Lagrange coefficients

$(a_1, \dots, a_n)$ $(b_1, \dots, b_n)$

Goal: Given additive shares $[a],[b]$, generate additive shares $[ab]$ using Shamir secret sharing.

⚠ $[ab] \neq (a_1 b_1, \dots, a_n b_n)$

Step 1 (additive to Shamir):

let's assume n is odd

this is party P_i's additive share of a

- Each party P_i picks a polynomial f_i of degree $\frac{n-1}{2}$ such that $f_i(0) = a_i$ and sends a share $f_i(j)$ to all other parties. (This is a Shamir secret sharing of a_i)
- The parties locally sum up their shares. They now have Shamir secret shares of a (and we do the same for b)

Proof: Party P_i's share is $\sum_{j=1}^n f_j(i) = (\sum_{j=1}^n f_j)(i) = F(i)$. This is a point on a polynomial $F(x)$ of degree $\frac{n-1}{2}$ and $F(0) = \sum_{j=1}^n f_j(0) = \sum_{j=1}^n a_j = a$.

Step 2 (multiplying Shamir shares):

- Each party P_i has shares $F(i)$, $G(i)$ where F, G are polynomials of degree $\frac{n-1}{2}$ and $F(0) = a$, $G(0) = b$
- Each party locally multiplies its shares: $\gamma_i = F(i) \cdot G(i)$
These are points on a polynomial $H(X) = F(x) \cdot G(x)$ of degree $n-1$ and $H(0) = F(0) \cdot G(0) = a \cdot b$

Thus, the parties now have a n-out-of-n Shamir secret sharing of $a \cdot b$

To get an additive sharing, we use Lagrange coefficients:

$$ab = H(0) = (V^{-1})_1 \cdot \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \sum_{i=1}^n \lambda_i \cdot H(i) = \sum_{i=1}^n \underbrace{\lambda_i \cdot F(i) \cdot G(i)}_{\text{this is party } P_i\text{'s additive share of ab}}$$

$$V^{-1} = \left(\begin{bmatrix} 1 & 1^1 & 1^2 & \dots & 1^{n-1} \\ 1 & 2^1 & 2^2 & \dots & 2^{n-1} \\ \vdots & & \ddots & & \vdots \\ 1 & n^1 & n^2 & \dots & n^{n-1} \end{bmatrix} \right)^{-1}$$

Wrapping up:

- If we use $\frac{n+1}{2}$-out-of-n Shamir secret sharing, we can multiply secret shared data and perform MPC.

- The number of corrupted parties must be less than $\frac{n+1}{2}$, otherwise the Shamir shares aren't private
  ($\frac{n+1}{2}$ colluding parties can reconstruct a and b)

As long as (strictly) more than $n/2$ parties are honest (an honest majority), the protocol is secure.

What if we want malicious security?

- Verifiable secret sharing
- Error correcting codes (Shamir secret sharing $\approx$ Reed Solomon codes!)

- We need $> 2/3$ of the parties to be honest
  [Ben-or - Goldwasser - Widgerson]

What if we want to support more corruptions? In particular, what happens when $n = 2$?

We need Crypto!

* Multiply shares using Oblivious Transfer or Somewhat Homomorphic Encryption
* For malicious security: add zk-proofs that each step of the protocol correctly followed the specification (+ some caveats) [Goldreich - Micali - Widgerson]