# Privacy Enhancing Technologies

## MPC

Florian Tramèr

So far:

Crypto primitives for specific problems

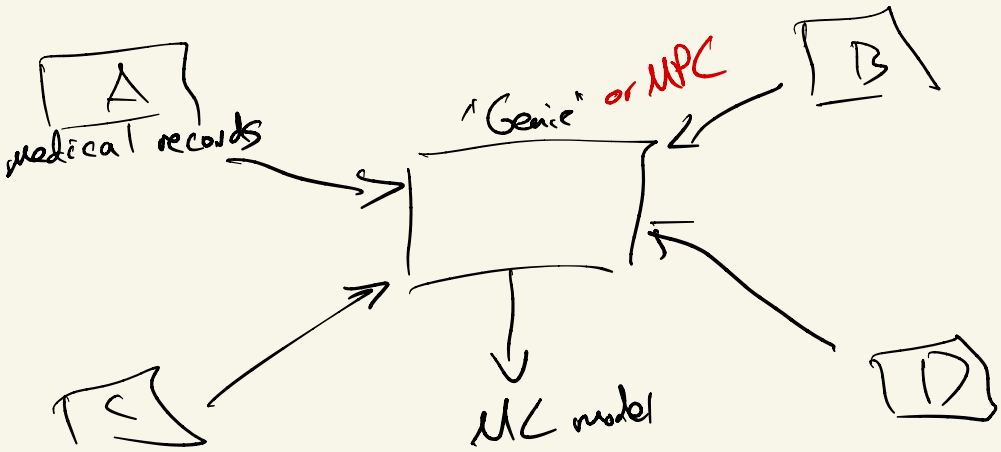
* commitments
* ZK proofs
* PIR, ORAM
* ...

Common: A way for mutually distrusting parties to compute some function over their data

Today: A way for MDPs to compute <u>any</u> function over their data

Main Result: "Any function that can be securely computed with the help of a trusted party, can also be securely computed without!" (assuming crypto)

Ex: Privak distributed ML




---

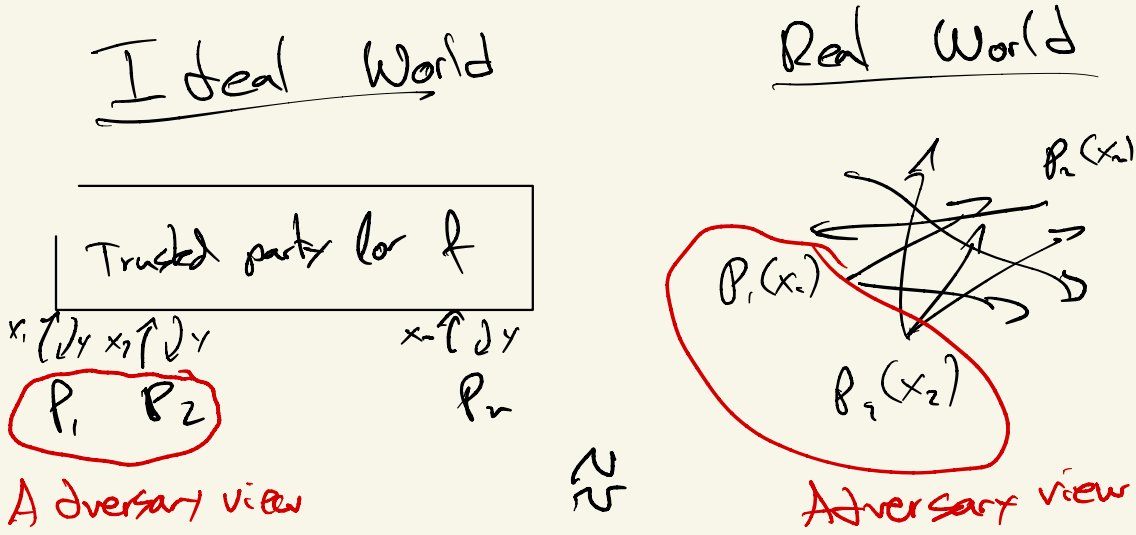
## Defining MPC

- $n$ parties $P_1, \dots, P_n$ with inputs $x_1, \dots, x_n$

  Compute $y \leftarrow f(x_1, \dots, x_n)$ ← public

- Adversary "corrupts" a subset of the parties and make them collude
  - ↳ learns the inputs of corrupted parties (choose them)
  - ↳ sees all msgs sent by and to corrupted parties

"Real ideal" paradigm / Simulation paradigm




$\overset{\mathcal{N}}{\approx}$

Formally: ← semi-honest MPC

there exists a simulator such that $\forall x_1, \ldots, x_n$ and all sets of corrupted parties $C \subseteq [n]$:

"minimum leakage of f"

$$Sim\left(C, \{x_i : i \in C\}, Y\right) \overset{\mathcal{N}^c}{\approx} \{View_i : i \in C\}$$

Adv view in ideal world — real corrupt view

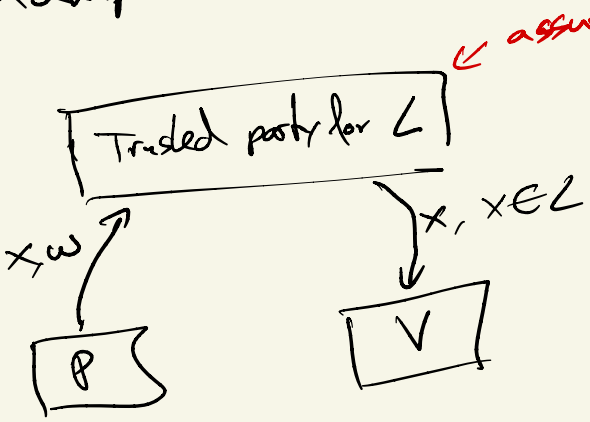
# On defining security:

"Two common ways"

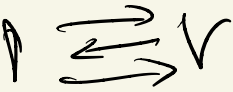
1) definition (game based)
   - ↳ security game (IND-CPA)
   - ↳ properties (soundness + ZK)

2) define an ideal functionality

Example: ZK proofs




Real world




---

Adversarial models:

1. Semi-honest security: Adv follows the protocol but they can do arbitrary other poly-time local compute to learn as much as possible about other parties' inputs

2. Malicious security: Adv can "do whatever it wants"

↳ defining security is hard

A blueprint for secure MPC:

1) build a semi-honest protocol
   ⇒ quite efficient and "simple"

2) add checks and balances

---

GMW protocol

Goldwasser Micali Wigderson (3x Turing Award winners..)

Setup:
- View $f$ as an arithmetic circuit

$$y \leftarrow C(x_1, \dots, x_n) \qquad x_i \in \mathbb{F}_p \quad y \in \mathbb{F}_p$$

↳ "+", "×" over $\mathbb{F}_p$

- Each party secret-shares their input with all others.

Additive secret sharing:

Share $\alpha \in \mathbb{F}_p$ among $n \geq 2$ parties:

$s_1, \ldots, s_{n-1} \xleftarrow{\$} \mathbb{F}_p$, $s_n = \alpha - \sum_{i=1}^{n-1} s_i$

We write: $[\alpha] = (s_1, \ldots, s_n)$, $\alpha = \sum_{i=1}^{n} s_i$

$\Rightarrow$ information theoretically, any subset of $n-1$ shares reveals nothing about $\alpha$

Proof: take $n-1$ shares, then their sum is $\alpha - s_j$ for the remaining share $s_j$ which is uniformly random in $\mathbb{F}_p$

---

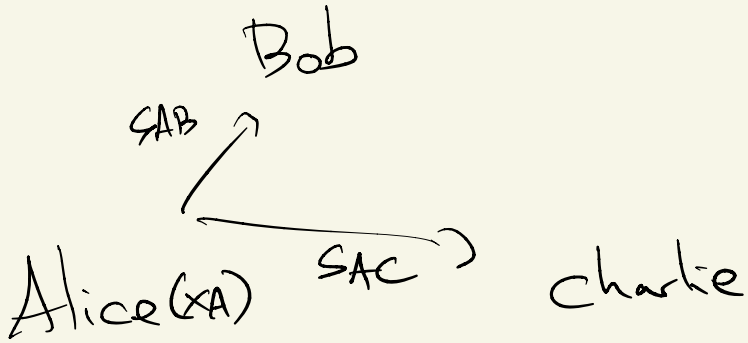
The protocol invariant: parties have secret shares of inputs to a gate $(+, \times)$ and compute shares of the output of the gate

## Protocol

1. Each party secret shares their input with all others
2. For an addition gate with inputs $[x], [y]$ the parties compute $[x+y]$
3. " multiplication gate " compute $[x \cdot y]$
4. All parties reveal their share of the output

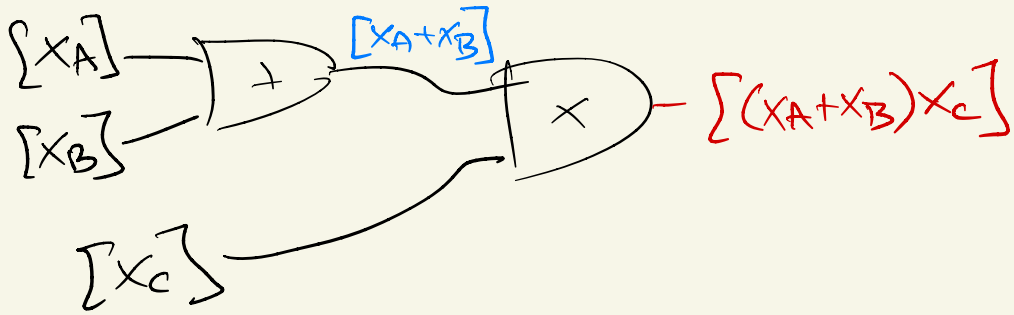
## Details:

1.




$[x_A] = (s_{AA}, s_{AB}, s_{AC})$

2/3.




For each wire, each party has one share of the value

## Addition

$$[x] + [y] = [x + y]$$

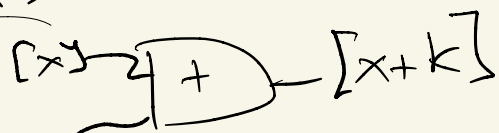
Why? $[x] = (x_1, \ldots, x_n)$, $x = \sum x_i$

$[y] = (y_1, \ldots, y_n)$, $y = \sum y_i$

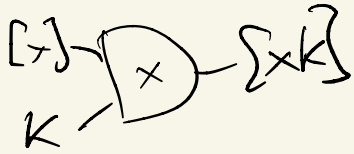
$$x + y = \sum x_i + \sum y_i = \sum_{i=1}^{n} (x_i + y_i)$$

$$[x+y] = (x_1+y_1, \ldots, x_n+y_n)$$

## Constants




$$[x+k] = (x_1+k, \ldots, x_n)$$




$$[xk] = (x_1k, \ldots, x_nk)$$

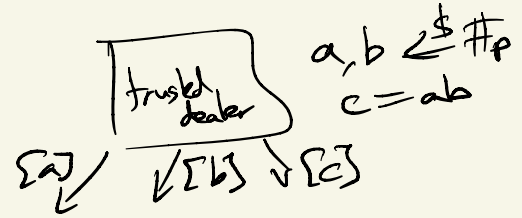
## Multiplication

$$[x] \cdot [y] = (x_1y_1, \ldots, x_ny_n)$$

$$\sum x_i y_i \neq \left(\sum x_i\right)\left(\sum y_i\right) = xy$$

$$x_1y_1 + x_1y_2 + x_1y_3 + \cdots$$

- Let's assume a "single" trusted party
  ↳ trusted dealer"




$P_1$ ... $P_n$

$(a_1, b_1, c_1)$ $(a_n, b_n, c_n)$

Neat trick (Beaver's trick): compute an arbitrary product $[xy]$ given shares of a random product $[ab]$

$$xy = ((x-a)+a)((y-b)+b)$$

$$\circledast = (x-a)(y-b) + a(y-b) + b(x-a) + ab$$

⚠ The parties have shares of $[a], [b]$ and $[ab]$ to compute $\circledast$ all they need is $(x-a)$ and $(y-b)$ "in the clear"

How can they get these, and is it secure?

Step 1: each party computes shares of $[b][x-a]$ and $[y-b]$ $= [\varepsilon]$

Step 2: (interaction) the parties all reveal their shares of $[x-a]$ and $[y-b]$

and compute $\delta \cdot \varepsilon = (x-a)(y-b)$

Step 3: recombine shares to get

$$[(x-a)(y-b)] + [a](y-b) + [b](x-a) + [ab]$$

$\Rightarrow$ only requires local compute

Why is this safe?

$x-a$ is a one time pad of $x$ with pad $a \in \$ \#_p$

AS LONG AS WE USE IT ONLY ONCE!

If the parties have $k$ shares of random products ("Beaver triples") $(a_i, b_i, c_i)$

where $k$ = # of mult. gates

then they can compute the whole circuit on secret-shared data.

Getting rid of the dealer

"slow", uses pk crypto

- Preprocessing phase:

  the parties do a protocol to generate many Beaver triples

  * independent of circuit inputs
  * independent of the circuit (except for # of $\times$ gates)

- Online phase: parties run the protocol we described

last, information theoretic

A modern approach: somewhat homomorphic encryption

rought:

* parties choose random shares $[a]$, $[b]$ and encrypt them
* Use FHE for the function $f(a_1, \dots, a_n, b_1, \dots, b_n) = (\sum a_i)(\sum b_i)$ we get an encryption of $c$
* run a "distributed decryption protocol" so each party gets a share of $c$

↳ specific MPC protocol

# Analysis

- Security:
  - in the online phase, any subset of $n-1$ parties learns nothing information theoretically except the output
  - in the offline phase, security comes from FHE scheme

- Efficiency: the computation per party is $O(|C|)$ where $|C|$ is # of gates in $C$

  the communication is linear in # of mult. gates and # of parties (unless there is a "public billboard")

# Malicious security

A malicious party can do lots of bad stuff

1) reveal incorrect output shares to bias the result
2) use incorrect share values at any gate

⋮

## Security properties (all captured by ideal functionality)

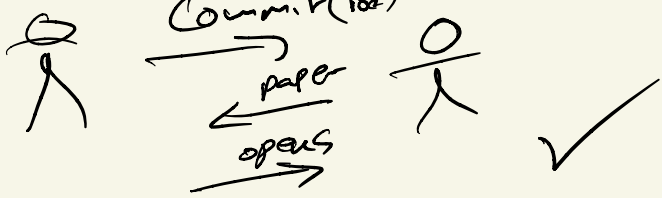
1) Privacy (Adv learns nothing except $y$)
2) Correctness (if one honest party outputs $y$, then $y = f(x_1, \ldots, x_n)$)
3) input independence (Adv cannot pick their input as a function of another user's input)
4) Fairness (if Adv learns $y$, then honest parties do too)

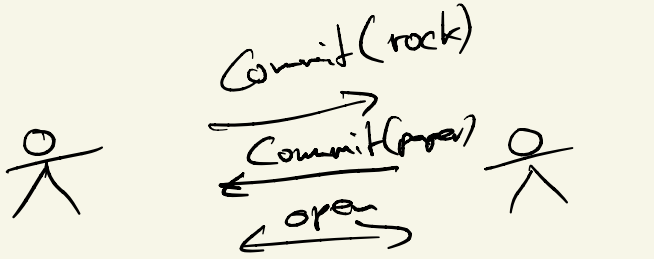
malicious secure MPC protocol for $n-1$ corrupt parties

impossible in general, possible if majority is honest

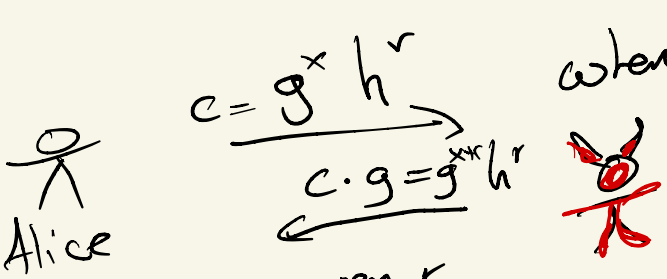
Ex: input independence

Rock paper scissors using commitments




example using Pedersen commitments




Issue: Pedersen is malleable (homomorphic)

Adversary wins $1/2$ of the time if Alice picks $x$ at random

# Beyond additive secret sharing

**Motivation:** In the GMW protocol, what happens if one party disconnects?

Answer: infinite sadness :(

We'd like a secret sharing where any $t \leq n$ shares are enough to reconstruct

↳ $t$-out-of-$n$ secret sharing

**Def:** A $t$-out-of-$n$ secret sharing over $\mathbb{Z}_p$ is a pair of efficient algs Gen and Recons

- $\text{Gen}(n, t, \alpha) \to (s_1, \ldots s_n)$ for $\alpha \in \mathbb{Z}_p$
- $\text{Recons}(s_{i_1} \ldots s_{i_t}) \to \alpha$

**Correctness:** for every $\alpha$, any subset of $t$ shares from $\text{Gen}(n, t, \alpha)$, Recons yields $\alpha$

**Perfect Privacy:** for any $\alpha, \alpha' \in \mathbb{Z}_p$ and $S \subseteq [n]$ of size $t-1$
$\text{Gen}(n, t, \alpha)[S] \equiv \text{Gen}(n, t, \alpha')[S]$

Shamir's secret sharing:

GS :- RSA

$\in \mathbb{Z}_p$

Gen$(n, t, \alpha)$: choose random coeffs $a_1, \dots, a_{t-1} \xleftarrow{\$} \mathbb{Z}_p$ and

define: $f(x) = \alpha + a_1 x + a_2 x^2 + \dots + a_{t-1} x^{t-1}$

$\Rightarrow f$ has degree $t-1$

$\rightarrow f(0) = \alpha$

shares: $S_i = (i, f(i)) \in \mathbb{Z}_p^2$

Recons$(S_{i_1}, \dots S_{i_t})$: recover $f(x)$ through interpolation

output $f(0)$

---

Correctness: Lemma: for any $d$ points $(x_1, y_1) \dots (x_d, y_d)$ where $x_i \neq x_j$ there is a unique polynomial of degree $d-1$ s.t. $f(x_i) = y_i$

**Proof**

$$f(x) = a_0 + a_1 x + \cdots + a_{d-1} x^{d-1}$$

we want:

$$f(x_1) = a_0 + a_1 x_1 + \cdots + a_{d-1} x_1^{d-1} = y_1$$
$$\vdots$$
$$f(x_d) = a_0 + a_1 x_d + \cdots + a_{d-1} x_d^{d-1} = y_d$$

in matrix form:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{d-1} \\ & & \vdots & & \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{d-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}$$

$\dim d \times d$ ← $V(x_1, \ldots, x_d)$ is Vandermonde matrix

"Fact": $\det(V(x_1, \ldots, x_d)) = \prod_{1 \le i < j \le d} (x_j - x_i)$

Fact: a matrix is invertible iff $\det \neq 0$

$\det(V) \neq 0$ iff all the $x_i$ are distinct

Fact: $V$ is invertible ∀

$$V \cdot \vec{a} = \vec{y}$$

$$\Leftrightarrow \quad \vec{a} = V^{-1} \vec{y}$$

$\hookrightarrow$ $\vec{a}$ is unique

Observation: we can interpolate $f$ using only linear operations on $V$

---

Privacy: let $S$ be a subset of $t-1$ parties

Let $\text{View}_S(x) = (y_1, \dots, y_{t-1})$

$\text{View}_S(x') = (y_1', \dots, y_{t-1}')$

We want: $\text{View}_S(x) \equiv \text{View}_S(x')$

Let $z_1, \dots, z_{t-1}$ be arbitrary values in $\mathbb{Z}_p$

$$\Pr_{a_1, \dots, a_{t-1}}\left[\text{View}_S(x) = (z_1, \dots, z_{t-1})\right] = \frac{1}{p^{t-1}}$$

$$\Pr_{\vec{a}}\left[\text{View}_S(x) = (z_1, \ldots, z_{t-1})\right]$$

$$= \Pr\left[(x, y_1, \ldots, y_{t-1}) = (x, z_1, \ldots, z_{t-1})\right]$$

$$= \Pr\left[V(0, x_1, \ldots x_{t-1}) \cdot \begin{pmatrix} x \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix} = (x, z_1, \ldots, z_{t-1})\right]$$

$$= \Pr_{\vec{a}}\left[(x, a_1, \ldots, a_{t-1}) = \underbrace{V^{-1}(x, z_1, \ldots, z_{t-1})}_{\text{some vector in } \mathbb{Z}_p^t}\right]$$

$$= \frac{1}{p^{t-1}}$$