Today - last lecture !

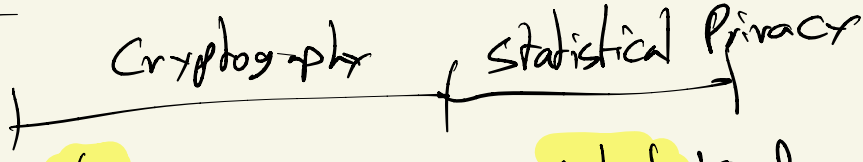
Monday : Find (see moodle)
↳ 2nd half of the course
(Concurrentization → MI)

This week: HW11 (check parts about MI)

next Tuesday : Merry Christmas !

⚠ Online feedback form

Recap



"How can we compute f without leaking anything"

"What kind of f can we compute without leaking too much"

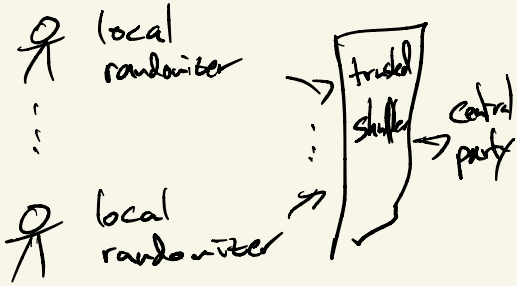
How to combine the two?

1) Decide "what" to compute
(ideally, something DP)

2) Decide "how" to compute it

A starting point: DP models

- Central model:
  - ⊖ trust a central party
  - ⊕ $\tilde{O}\left(\frac{\sqrt{k}}{\varepsilon n}\right)$ error for k counts
- Local model:
  - ⊕ no trust
  - ⊖ $\tilde{O}\left(\frac{1}{\varepsilon\sqrt{n}}\right)$ error
- Shuffle model:
  "central model with anonymity"



- Shuffle :
  - ⊕ error similar to central model (up to logarithmic factors)
  - ⊖ we still need a trusted shuffler
    - ↳ idea: this is easier to implement (with crypto)

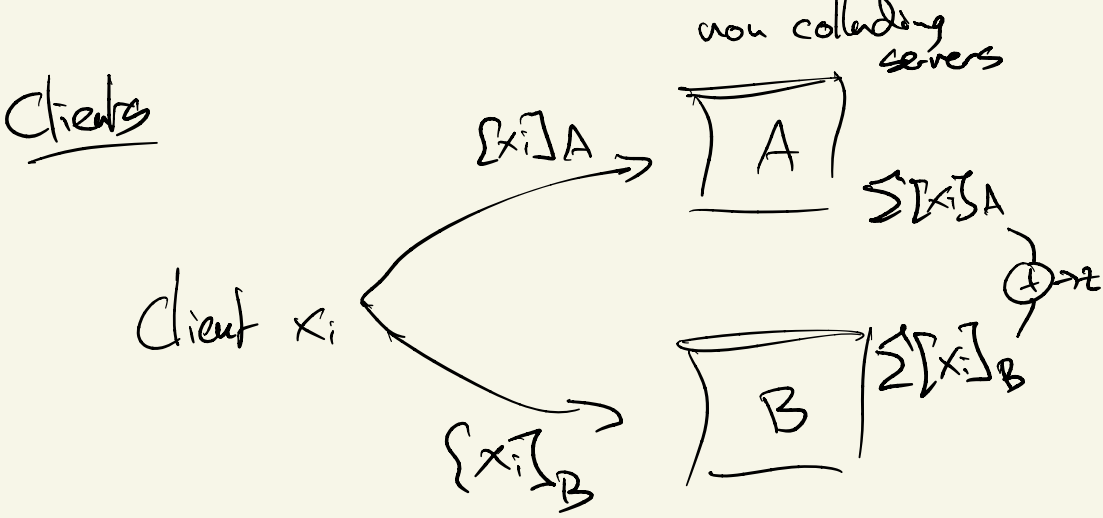
- MPC model (replace the trusted curator by MPC)
  - ⊕ equivalent to central model (up to cryptographic indist.)
  - ⊖ extremely expensive

- Distributed trust model (MPC with non-colluding servers)

Example: PRIO

Goal: private aggregation

- n parties with inputs $x_i \in \mathbb{F}_p^d$
- we want to compute $\sum_{i=1}^{n} x_i$ (+ noise)



* Correct !
* Cryptographically private
* Efficient

Integrity Problem: one client can mess up everything by sending garbage

Solution: ZK-proofs for specific properties checks (e.g. $x_i = \{1, 0, \dots, 0\}$)
There are very efficient Proofs

(SNARKs, (linear)-PCPs, ...)

What about DP?

"Distributed Noise Generation"

- "non-malicious": assume m parties, at least t of which will add noise.

$\Rightarrow$ make sure that the t parties add enough noise

- if you want noise $N(0, \sigma^2)$, each honest party adds $N(0, \frac{\sigma^2}{t})$ noise
  - if t parties are honest $\Rightarrow N(0, \sigma^2)$
  - if $m \geq t$ parties are honest $\Rightarrow N(0, \frac{m}{t}\sigma^2)$

Example in PRIO: each server adds $N(0, \sigma^2)$ noise

- malicious case: MPC protocols for noise generation

Example: Dwork et al.

insight: $N(0, \sigma^2) \approx \text{Binom}(n, p)$

$= \sum_n \text{Bernoulli}(p)$

Protocol: parties generate shares of random bits $[b_i]$ and then add them

- A party could cheat

1) send shares of $z \notin \{0,1\}$

Solution: cheap ZK proofs

2) send shares of bits with wrong bias

Cute trick: (say we want $p = 1/2$)

1) everyone generates shares of bits
2) we open $b(\lambda)$ bits $b_1, \ldots, b_{b(\lambda)}$
3) $c_1, \ldots, c_\ell \leftarrow PRG(b_1, \ldots, b_{b(\lambda)})$
4) use public bits to bias the secret shared ones

$$[b_i] \rightarrow [b_i \oplus c_i]$$

these are unknown and unbiased!!