Locally Private
Heavy Hitters

Uri Stemmer

Based on a work with

Raef Bassily, Kobbi Nissim, Abhradeep Thakurta (NIPS'17)

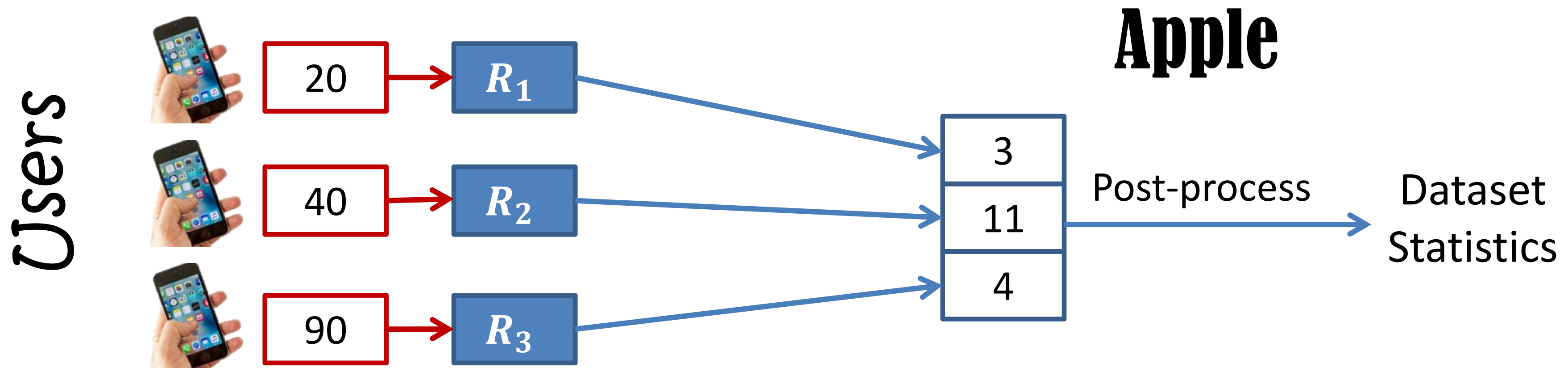
and a work with

Mark Bun and Jelani Nelson (PODS'18)

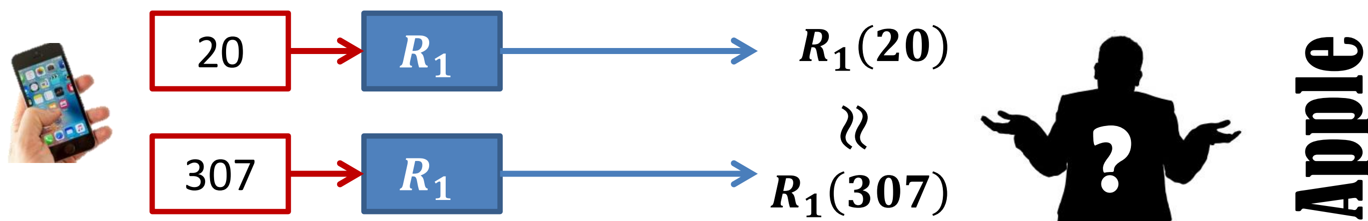
What is Local Differential Privacy?

[DMNS'06] [KLNRS'08]

Users retain their data and only send the server randomizations which are safe for publication



Definition – Local Differential Privacy (simplified)
• ϵ-LDP algorithm accesses every data entry only once, via an ϵ-local randomizer • ϵ-local randomizer is an algorithm $R: X \to Y$ s.t. $\forall x, x' \in X, \ \forall y \in Y$ $\Pr[R(x) = y] \leq e^{\epsilon} \cdot \Pr[R(x') = y]$



Why use LDP?

- Valuable information about users while providing **strong privacy guarantees**
- Privacy preserved even if the organization is subpoenaed
- Reduces organization liability for securing the data

Challenges

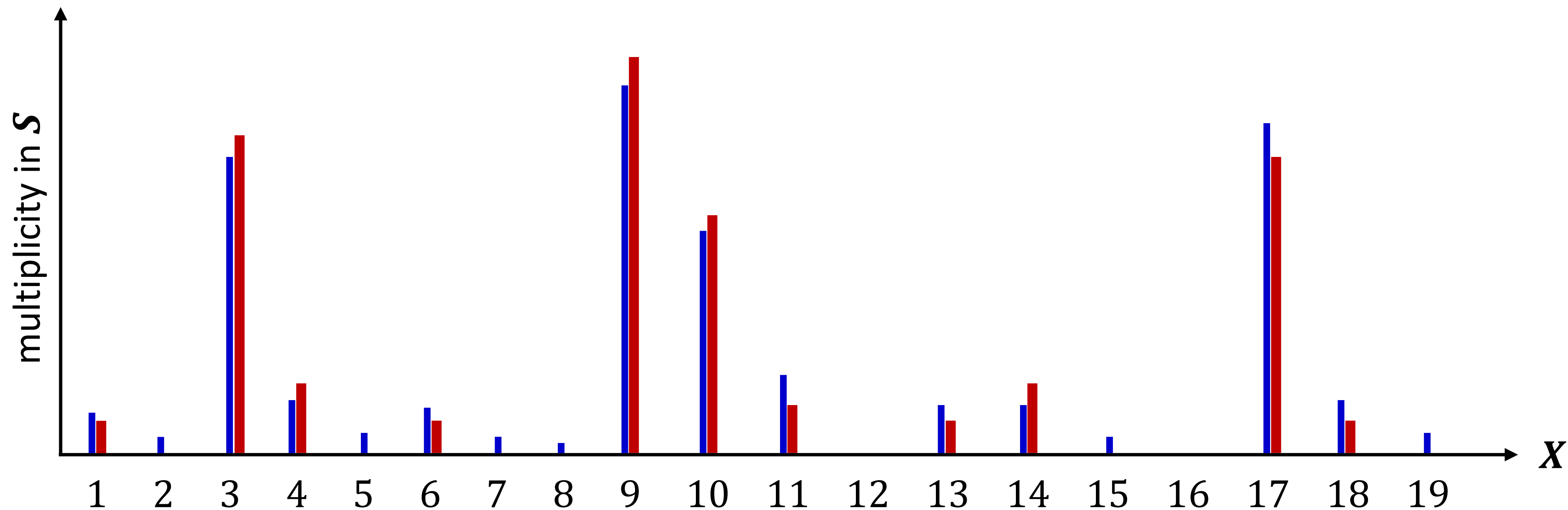
- As every user randomizes her data, accuracy is reduced
- **Number of users might be very large (in the millions)**
- Optimizing runtime and memory usage becomes crucial

Problem Statement: Histograms

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_S$ s.t. $\max_{x \in X} \left| \hat{f}_S(x) - f_S(x) \right|$ is small

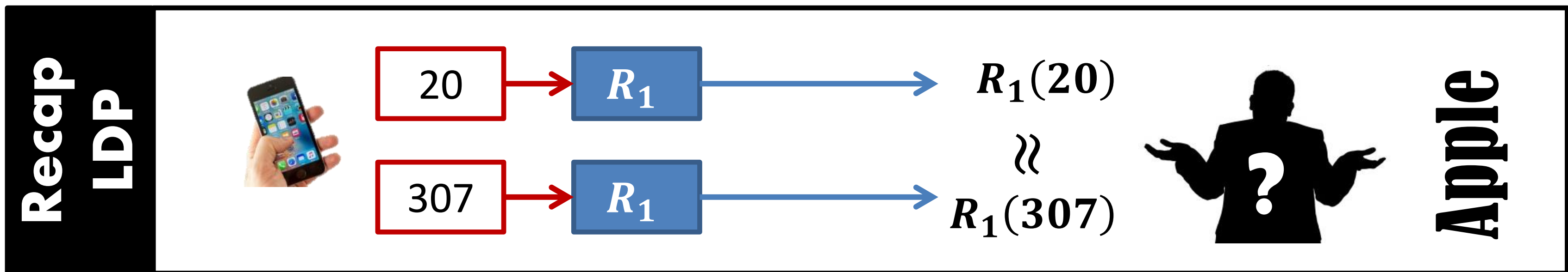


Problem Statement: Histograms

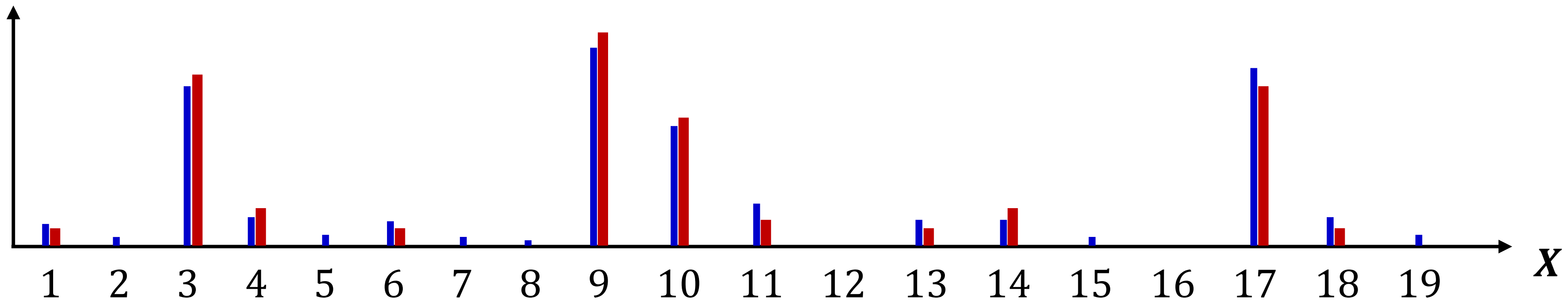
- Distributed database $S = (x_1, \ldots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_S$ s.t. $\max_{x \in X} \left| \hat{f}_S(x) - f_S(x) \right|$ is small



Apple can learn that many users hold '17', without knowing which are these users!



Why solve under LDP?

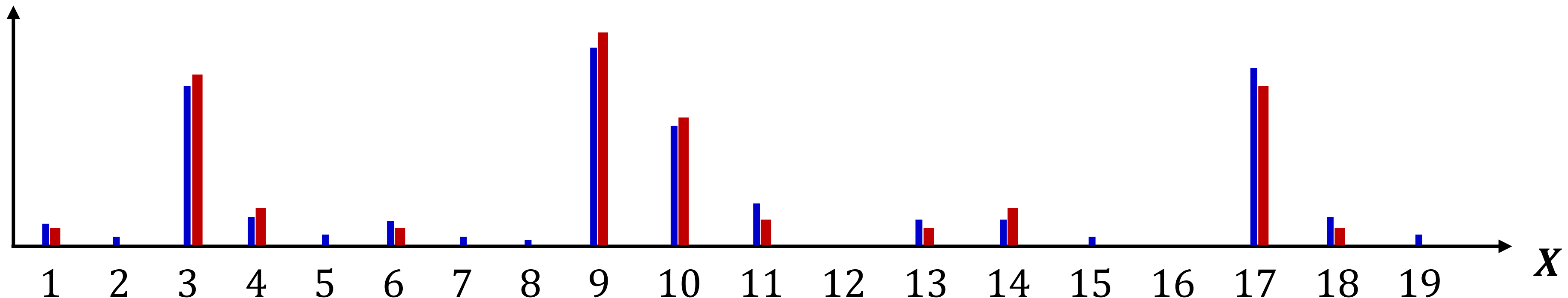
- Distributed database $S = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

- Arguably the most well-studied problem under LDP, Important subroutine for solving many other problems

  [MS 06], [HKR 12], [EP 14], [BS 15], [QYYKXR 16], [TVVKFSD 17]…

- **Google** and **Apple** are using LDP algorithms for this problem in the **Chrome browser** and in **iOS-10:**
  - QuickType suggestions, Emoji suggestions, Lookup Hints, Energy Draining Domains, Autoplay Intent Detection, Crashing Domains, Health Type Usage

- Most widespread industrial application of differential privacy to date



Frequency Oracles

- Distributed database $\boldsymbol{S} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n) \in \boldsymbol{X}^n$, where user $\boldsymbol{i}$ holds $\boldsymbol{x}_i \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f}_{\boldsymbol{S}}(\boldsymbol{x})$

Observe: If $|\boldsymbol{X}|$ is large, then efficient algorithms cannot output estimations for every $\boldsymbol{x} \in \boldsymbol{X}$ directly.

Simplified Goal – Frequency Oracle:

Frequency oracle is an algorithm that, after communicating with the users, outputs a *data structure* capable of approximating $\boldsymbol{f}_{\boldsymbol{S}}(\boldsymbol{x})$ for every $\boldsymbol{x} \in \boldsymbol{X}$

Our Results*

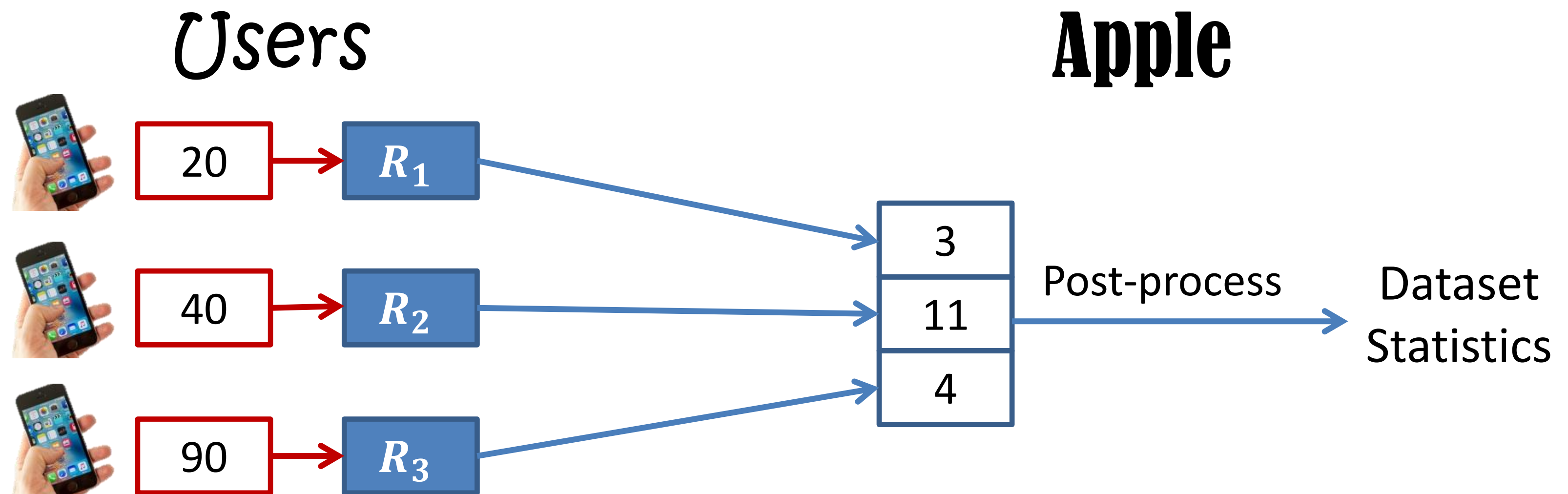
LDP Frequency oracle with error $\lesssim \boldsymbol{O}(\sqrt{\boldsymbol{n}})$

- **Server:** instantiation time $\tilde{\boldsymbol{O}}(\boldsymbol{n})$, memory $\tilde{\boldsymbol{O}}(\sqrt{\boldsymbol{n}})$, query time $\tilde{\boldsymbol{O}}(\boldsymbol{1})$
- **Users:** runtime, communication, and memory $\tilde{\boldsymbol{O}}(\boldsymbol{1})$

Improves on the algorithm of Apple where instantiation time is $\tilde{\boldsymbol{O}}(\boldsymbol{n}^{1.5})$

STEP BACK:

How can LDP be useful at all?



LDP Warmup: Oracle for X={±1}

- Distributed database $S = (x_1, \ldots, x_n) \in \{\pm 1\}^n$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $f_S(1)$

Local randomizer – randomized response $R(x)$:

Return x w.p. $\approx \frac{1}{2} + \epsilon$ (and -x otherwise)

Observe: Any output (±1) is almost as equally likely to result from any input (±1)

Protocol: From every user i obtain $y_i \leftarrow R(x_i)$. Return $\frac{1}{4\epsilon}\left(2\epsilon n + \sum_i y_i\right)$

Analysis: $\mathbb{E}\left[\sum_i y_i\right] = \sum_{i:x_i=1} \mathbb{E}[y_i] + \sum_{i:x_i=-1} \mathbb{E}[y_i]$

$(-1) \cdot \left(\frac{1}{2} + \epsilon\right) + 1 \cdot \left(\frac{1}{2} - \epsilon\right) = -2\epsilon$

$$= f_S(1) \cdot 2\epsilon - f_S(-1) \cdot 2\epsilon = f_S(1) \cdot 4\epsilon - n \cdot 2\epsilon$$

Hoeffding bound:

w.h.p. our estimation error is at most $\approx \frac{1}{\epsilon}\sqrt{n}$

STEP FORWARD

Frequency Oracle

when the domain is large

Frequency Oracle for large domains

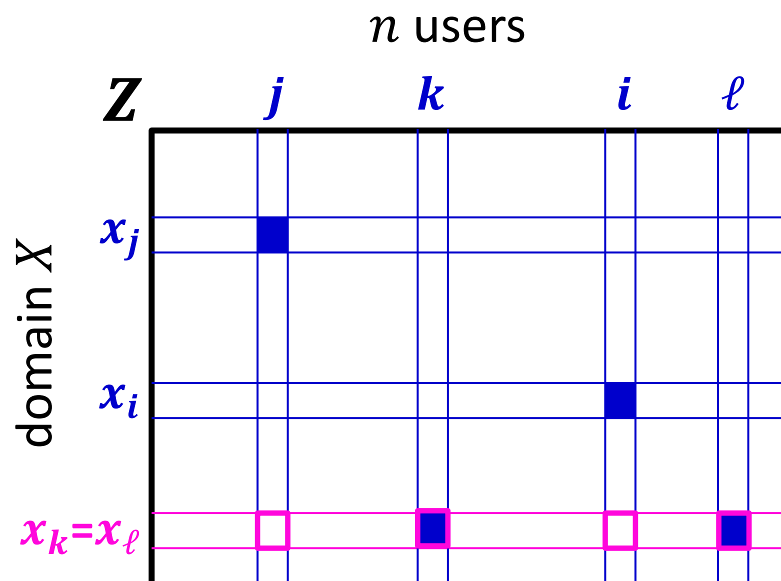
Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm \mathbf{1}\}^{|X| \times n}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$

Users randomize their corresponding bits:

User $\boldsymbol{i}$ sends $\boldsymbol{y_i} = \boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} + \epsilon$

$\boldsymbol{y_i} = -\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} - \epsilon$



Server: Given a query $\boldsymbol{x} \in \boldsymbol{X}$, return $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \frac{1}{2\epsilon} \sum_{i \in [n]} \boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$

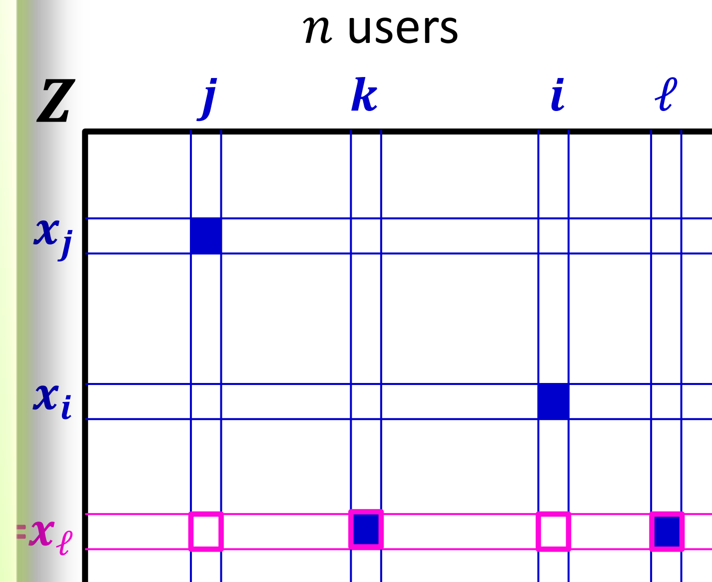
$$\mathbb{E}\left[\sum_{i \in [n]} \boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]\right] = \sum_{i : x_i = x} \mathbb{E}[\boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]] + \sum_{i : x_i \neq x} \mathbb{E}[\boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]] = 2\epsilon \cdot \boldsymbol{f_S}(\boldsymbol{x})$$

Hoeffding bound: w.h.p. our estimation error is at most $\approx \frac{1}{\epsilon}\sqrt{\boldsymbol{n} \cdot \log|\boldsymbol{X}|}$

arge domains

Optimizations:

1) Every column individually contains pairwise independent bits
2) We aim for error $\approx \sqrt{n}$
   $\Longrightarrow$ Can hash X into range of size $\approx \sqrt{n}$
3) So every column consists of $\approx \sqrt{n}$ pairwise bits (takes $\approx \log \sqrt{n}$ bits)
   $\Longrightarrow$ At most $\approx \sqrt{n}$ different columns
4) Computing $\hat{f}(x)$ requires summing only $\approx \sqrt{n}$ elements, so computing all possible answers takes $\approx n$ time



$= \frac{1}{2\epsilon} \sum_{i \in [n]} y_i \cdot Z[x, i]$

$$\mathbb{E}\left[\sum_{i \in [n]} y_i \cdot Z[x,i]\right] = \sum_{i: x_i = x} \mathbb{E}[y_i \cdot Z[x,i]] + \sum_{i: x_i \neq x} \mathbb{E}[y_i \cdot Z[x,i]] = 2\epsilon \cdot f_S(x)$$

Hoeffding bound: w.h.p. our estimation error is at most $\approx \frac{1}{\epsilon}\sqrt{n \cdot \log|X|}$

Summary

- Efficient LDP frequency oracle protocol
- In the paper we extend our protocol to identify all "heavy-hitters" in the data (which are the frequent input elements)
- Our protocols obtain:
  - **Optimal error**
  - **$\tilde{O}$ptimal runtime, memory, communication**

Questions?