

Locally Private Heavy Hitters

Uri Stemmer

Based on a work with

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

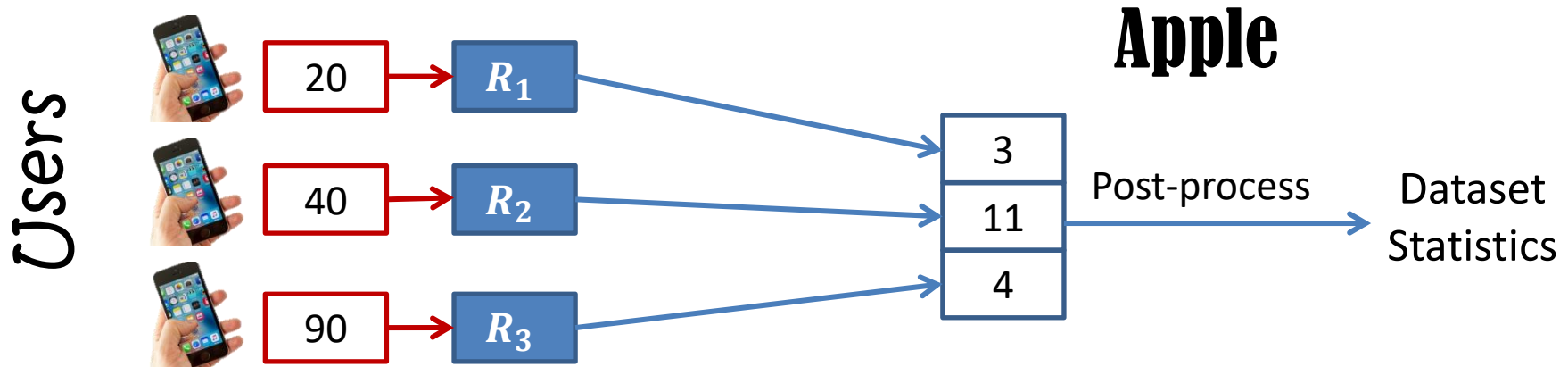
and a work with

Mark Bun and Jelani Nelson (PODS'18)

What is Local Differential Privacy?

[DMNS'06]
[KLNRS'08]

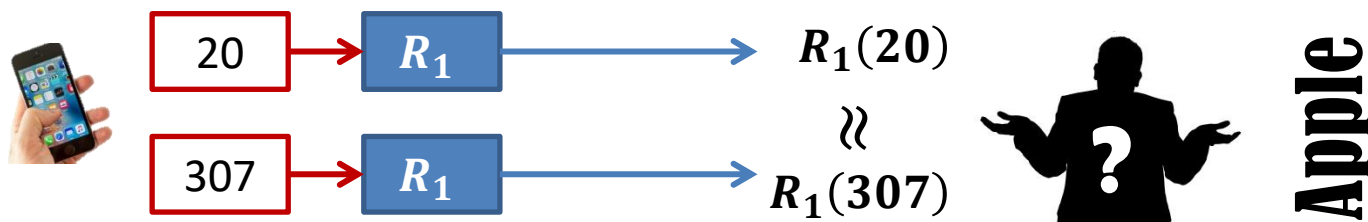
Users retain their data and only send the server randomizations which are safe for publication



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 $R: X \rightarrow Y$ 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

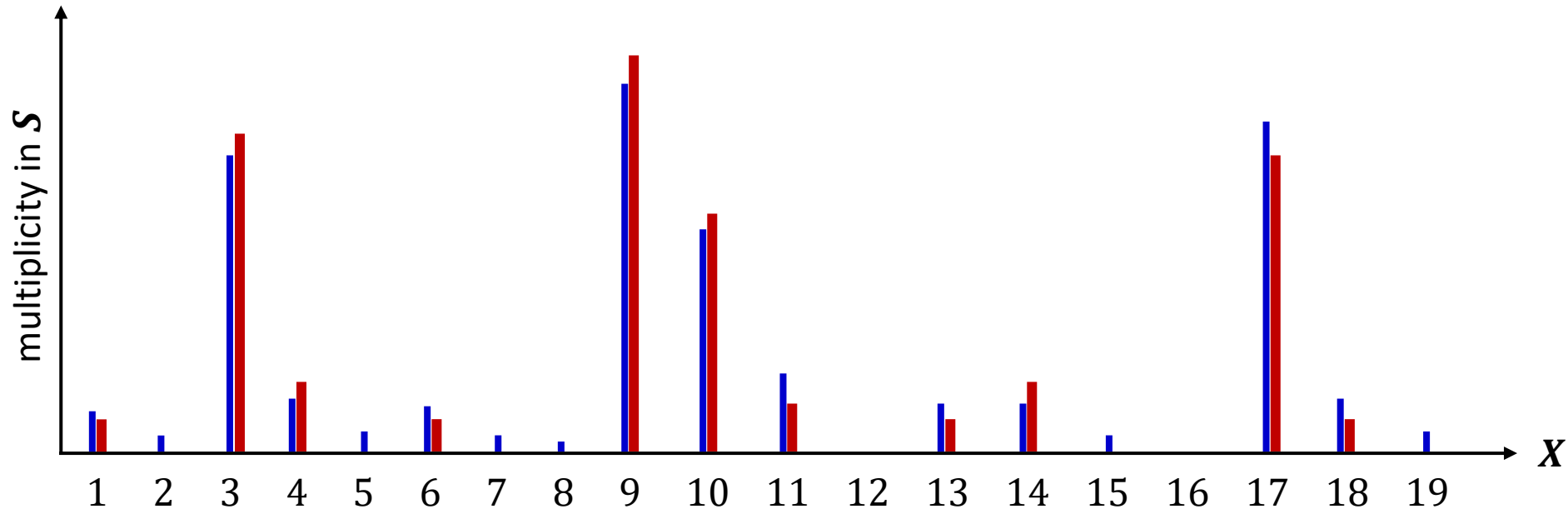
- As every user randomizes her data, accuracy is reduced
- **Number of users might be very large (in the millions)**
- Optimizing runtime and memory usage becomes crucial

Problem Statement: Histograms

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

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_{\mathcal{S}}$ s.t. $\max_{x \in X} |\hat{f}_{\mathcal{S}}(x) - f_{\mathcal{S}}(x)|$ is small

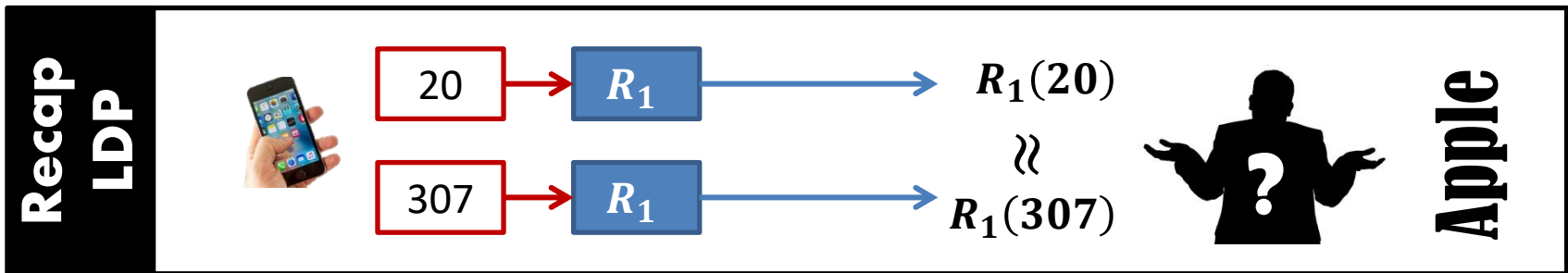


Problem Statement: Histograms

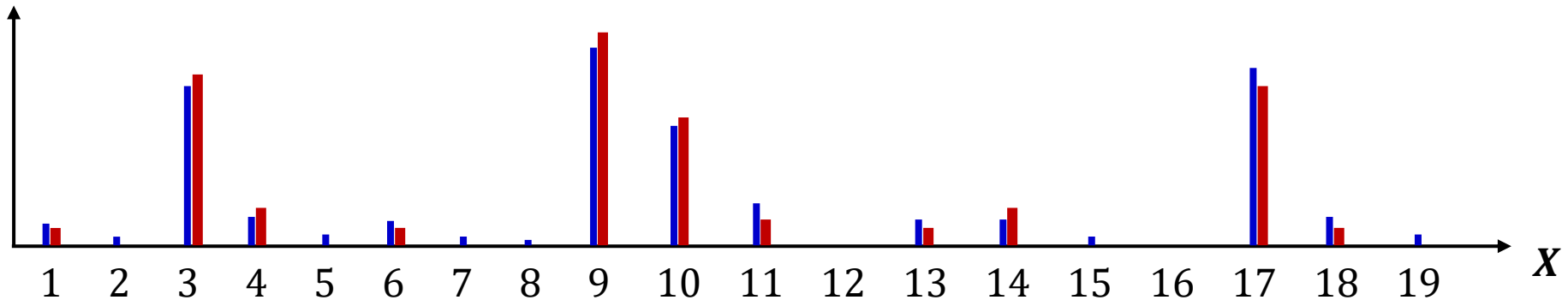
- Distributed database $\mathcal{S} = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in \mathcal{S} , denoted $f_{\mathcal{S}}(x)$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_{\mathcal{S}}$ s.t. $\max_{x \in X} |\hat{f}_{\mathcal{S}}(x) - f_{\mathcal{S}}(x)|$ is small



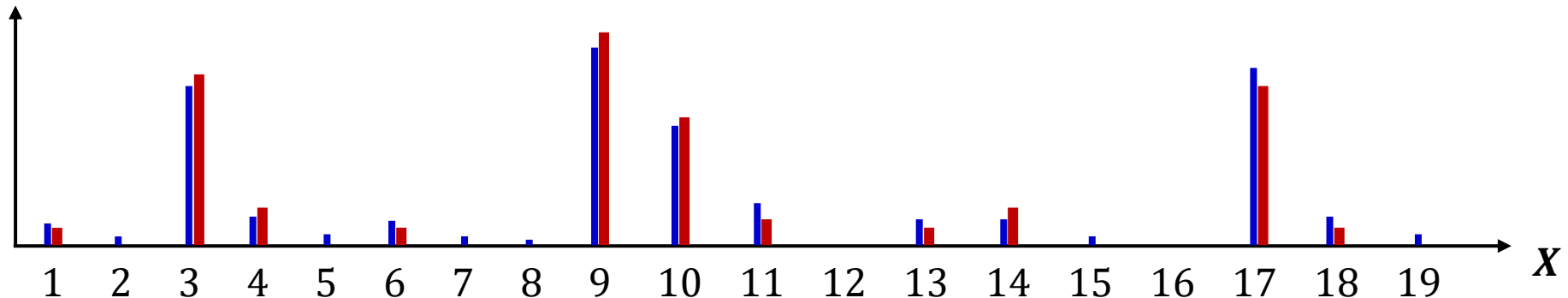
Apple can learn that many users hold '17', without knowing which are these users!



Why solve under LDP?

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

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

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

Simplified Goal – Frequency Oracle:

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

Our Results*

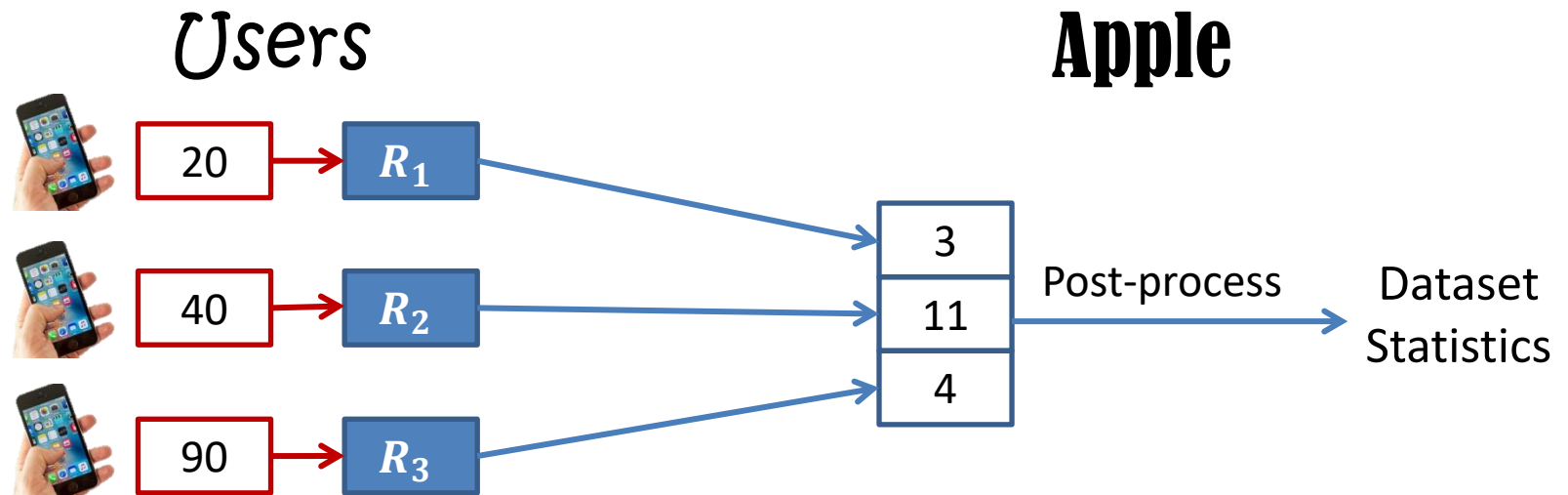
LDP Frequency oracle with error $\lesssim O(\sqrt{n})$

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

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

STEP BACK:

How can LDP be useful at all?



LDP Warmup: Oracle for $\mathbf{X}=\{\pm 1\}$

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

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 $\frac{1}{4\epsilon} (2\epsilon n + \sum_i y_i)$

Analysis:
$$\mathbb{E}[\sum_i y_i] = \sum_{i:x_i=1} \mathbb{E}[y_i] + \sum_{i:x_i=-1} \mathbb{E}[y_i]$$
$$(-1) \cdot \left(\frac{1}{2} + \epsilon\right) + 1 \cdot \left(\frac{1}{2} - \epsilon\right) = -2\epsilon$$
$$= f_{\mathcal{S}}(\mathbf{1}) \cdot 2\epsilon - f_{\mathcal{S}}(-\mathbf{1}) \cdot 2\epsilon = f_{\mathcal{S}}(\mathbf{1}) \cdot 4\epsilon - n \cdot 2\epsilon$$

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

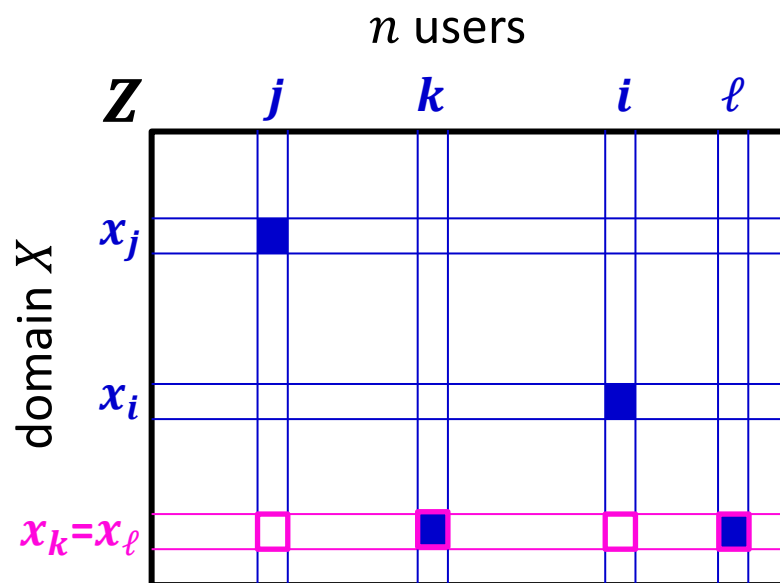
Setting:

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

Users randomize their corresponding bits:

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

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



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

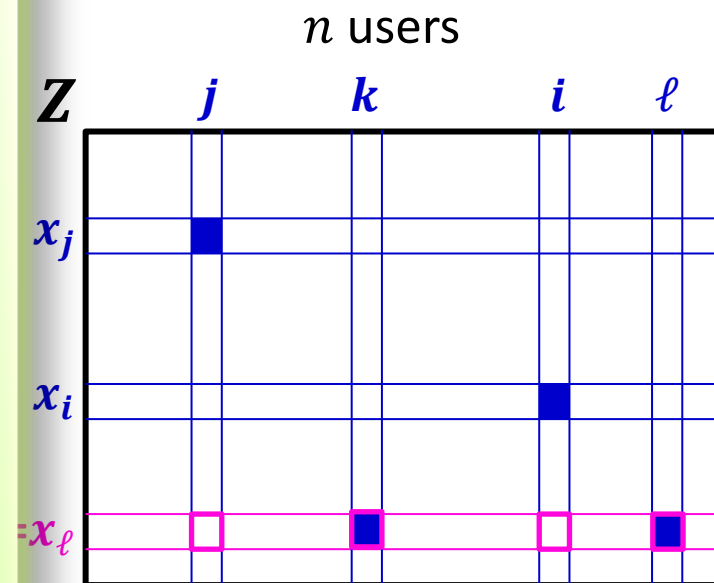
$$\mathbb{E} \left[\sum_{i \in [n]} y_i \cdot Z[x, i] \right] = \sum_{i: x_i = x} \mathbb{E}[y_i \cdot Z[x, i]] + \sum_{i: x_i \neq x} \mathbb{E}[y_i \cdot Z[x, i]] = 2\epsilon \cdot f_S(x)$$

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

Optimizations:

- 1) Every column individually contains pairwise independent bits
- 2) We aim for error $\approx \sqrt{n}$
 \Rightarrow Can hash \mathbf{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)
 \Rightarrow At most $\approx \sqrt{n}$ different columns
- 4) Computing $\hat{\mathbf{f}}(\mathbf{x})$ requires summing only $\approx \sqrt{n}$ elements, so computing all possible answers takes $\approx n$ time

Large domains



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

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

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?
Questions?