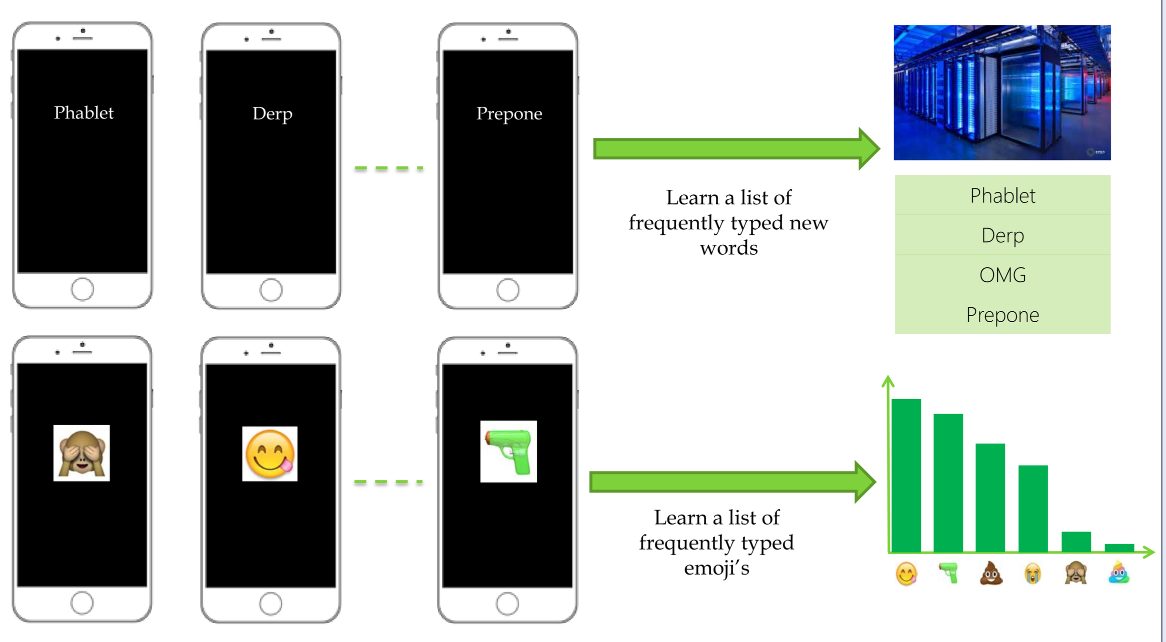
Practical Locally Private Heavy Hitters

Raef Bassily[1] Kobbi Nissim[2] Uri Stemmer[3] Abhradeep Thakurta[4]

[1]Ohio State University [2]Gerogetown University [3]Harvard University [4]University of California Santa Cruz

Learning from private data

Example: Learn new words and emoji's

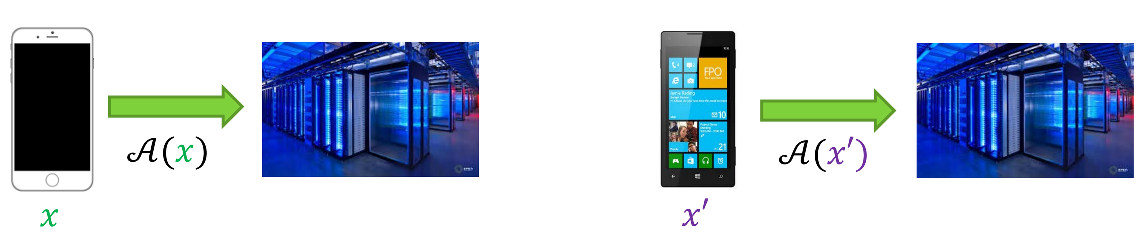


Problem definition

- Consider a set of data samples distributed across devices
  - Each data sample corresponds to one device

- Want: Learn *frequency statistics* and *heavy hitters* while preserving *local differential privacy*
  - Heavy hitters: All elements in the domain with frequency higher than a fixed threshold

Local differential privacy [Warner 65, EGS 03, DMNS 06, KLNRS 08]



Intuition: Every possible input x is plausible given transcript $\mathcal{A}(x)$. Need not trust a centralized data curator.

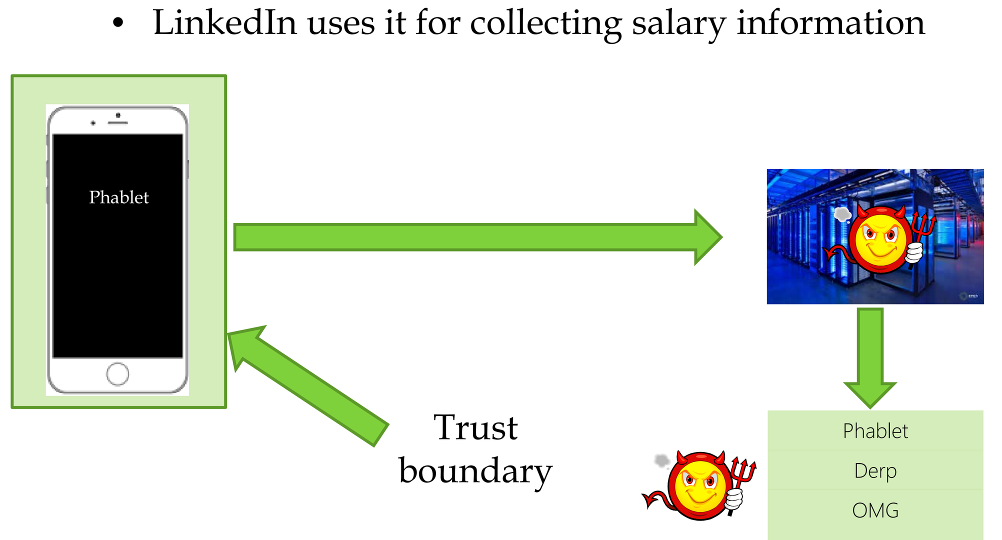
A randomized algorithm $\mathcal{A}$ is ϵ – locally differentially private if
- for all possible data samples x, x' from the domain
- for all (measurable) sets of answers E

$$\Pr[\mathcal{A}(x) \in E] \leq e^{\epsilon} \Pr[\mathcal{A}(x') \in E]$$

Why is privacy a concern?

- Adversary can *observe the output*, or even worse, *compromise the server*

- Companies use similar technology for analyzing *very sensitive data*
  - Google uses it for collecting malicious URL visit
  - Apple uses it for collecting new words people type, health related information
  - LinkedIn uses it for collecting salary information



Our contributions

Two new algorithms with optimal error, with optimal time, space and communication complexity

- TreeHist: Based on classic count sketch [CCF02]
  - Nearly optimal error guarantee
  - Easily implementable in practice

- Bitstogram: Based on error correcting codes
  - Optimal error guarantee

Implementation of our TreeHist protocol
- Scales to 10 million data samples, with domain $\approx 10^8$
- Performs better than state-of-the-art open source tools

Our results at a glance

Performance metric	TreeHist (this work)	Bitstrogram (this work)	Bassily and Smith'15
Server time	$\tilde{O}(n)$	$\tilde{O}(n)$	$\tilde{O}(n^{5/2})$
User time	$\tilde{O}(1)$	$\tilde{O}(1)$	$\tilde{O}(n^{3/2})$
Server memory	$\tilde{O}(\sqrt{n})$	$\tilde{O}(\sqrt{n})$	$\tilde{O}(n^2)$
User memory	$\tilde{O}(1)$	$\tilde{O}(1)$	$\tilde{O}(n^{3/2})$
Communication/ user	$\tilde{O}(1)$	$\tilde{O}(1)$	$\tilde{O}(1)$
Error	$O\left(\sqrt{n \log n \log d}\right)$	$O\left(\sqrt{n \log d}\right)$	$O\left(\sqrt{n \log d}\right)$

n: Number of data samples
d: Domain size
Error η: Can find all heavy hitters with frequency $\geq 3\eta$, and none with frequency $< \eta$

(Table hides terms in ϵ)

Prior work

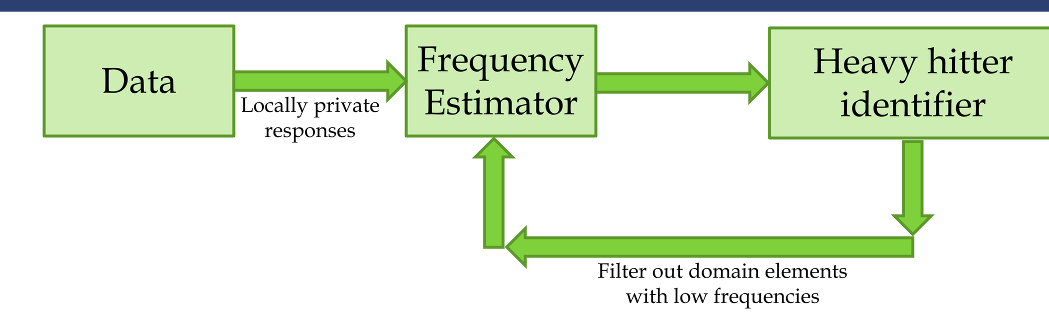
[Bassily and Smith 2015]
- Algorithm based on dimensionality reduction via the Johnson Lindenstrauss transformation
- Optimal error guarantee, non-optimal in terms of computation, storage and communication

[Thakurta et al., 2016]
- Optimal frequency estimator via count-median-sketch
- Heavy hitters algorithm is a heuristic, with no formal guarantees

[Erlingsson et al., 2013, Fanti et al., 2016]
- Empirical results only, with no formal guarantees

[Hsu et al., 2014]
- Algorithm based on group testing

Algorithm overview



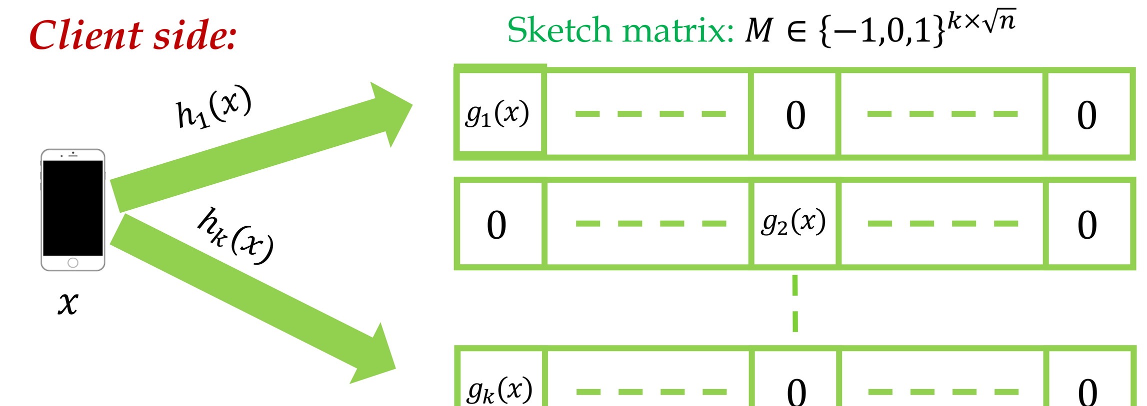
Locally private frequency estimator

Problem: Design locally differentially private estimator with

Accuracy: $O(\sqrt{n \log d})$ Query time / estimate: $O(\log d)$
Communication: 1 bit/ user Client computation: Constant
Server computation: $O(\sqrt{n}\log d)$

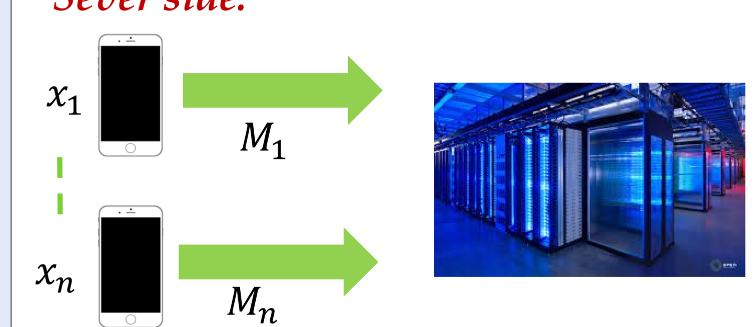
Non-private count-sketch [CCF02]

Client side: Sketch matrix: $M \in \{-1,0,1\}^{k \times \sqrt{n}}$



Hash functions: Pairwise independent s.t. $\forall i \in [k], h_i: [d] \to [\sqrt{n}]$
Pairwise independent s.t. $\forall i \in [k], g_i: [d] \to \{\pm 1\}$

Sever side:



Sketch: $S = \sum_{i=1}^{n} M_i$

Frequency estimate of x: Median$\{g_i(x) S[i, h_i(x)]\}$

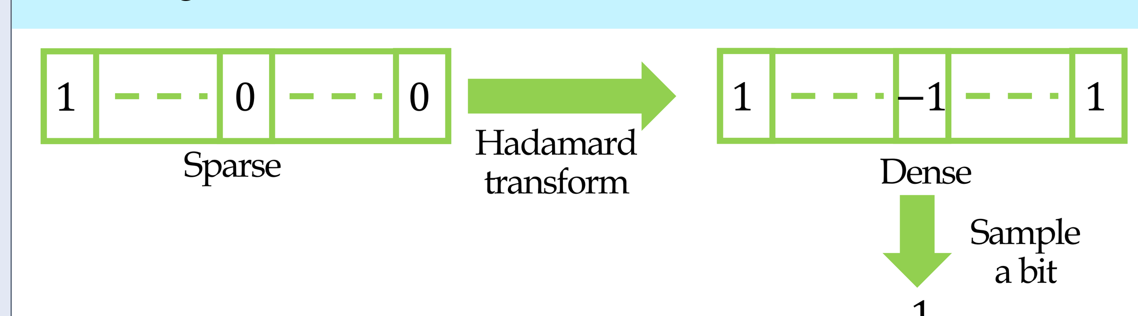
Differentially private count sketch

Client side: 1. Sample a row M_i from the sketch matrix M
2. Replace each zero of M_i with ± 1 sampled w.p. = ½
3. Flip each bit of M_i with probability $\frac{e^{\epsilon}}{1+e^{\epsilon}}$, and send (i, M_i)

Server side: For client report (i, v), add $\left(\log d \times \frac{e^{\epsilon}+1}{e^{\epsilon}-1}\right) \cdot v$ to the i-th row of the server side sketch S

Theorem: The algorithm is ϵ locally differentially private, and satisfies all the desired utility guarantees, *except communication*

Reducing communication via Hadamard transform

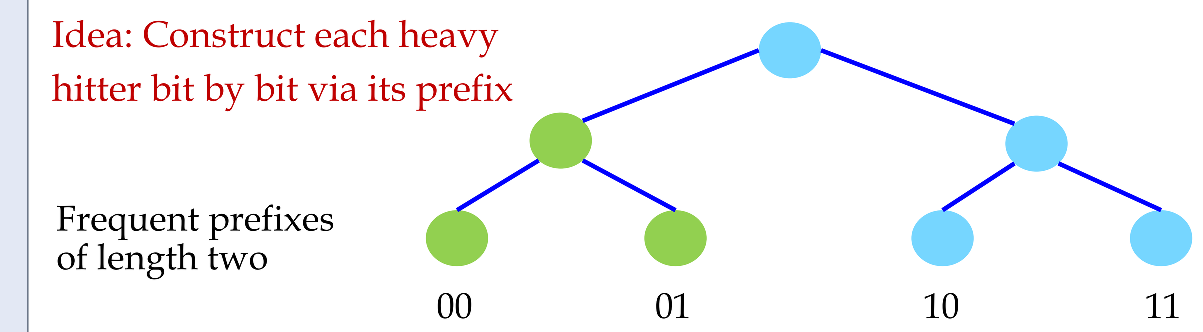


Theorem: Hadamard transform and sampling does not affect privacy or utility, but *reduces communication to $\tilde{O}(1)$ bit*

(Simplified) TreeHist protocol

Observation: Any heavy hitter can be represented using $\log d$ bits

Idea: Construct each heavy hitter bit by bit via its prefix



Observation: If a domain element has frequency $\geq \sqrt{n}$, so does all its prefixes

Observation: Roughly $\sqrt{n}$ elements per level of the tree

Algorithm: Each client samples a random prefix, and sends it via the locally differentially private count sketch protocol

The server computes the frequent heavy hitters via the prefix tree

(Simplified) Bitstogram protocol

- Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$ where $\boldsymbol{x_i} \in [\boldsymbol{d}]$ is the data of user $\boldsymbol{i}$

- Assume we have frequency oracle $\boldsymbol{\mathcal{O}}$ with error $\boldsymbol{\tau}$

- Public randomness: Hash function $\boldsymbol{h}: \boldsymbol{X} \to [\boldsymbol{T}]$
  Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\boldsymbol{\mathcal{O}}$ on hash range. But how?

- Fix a "heavy-hitter" $\boldsymbol{x}^*$ appearing $\boldsymbol{2\tau}$ times in the data $\boldsymbol{S}$.
  Denote $\boldsymbol{t}^* = \boldsymbol{h}(\boldsymbol{x}^*)$.
  For simplicity: assume that only $\boldsymbol{x}^*$ maps into $\boldsymbol{t}^*$ (from $\boldsymbol{S}$)

- For $\ell \in [\log \boldsymbol{d}]$ define $\boldsymbol{S}_\ell = (\boldsymbol{h}(\boldsymbol{x_i}), \boldsymbol{x_i}[\ell])_{i \in [n]}$, where $\boldsymbol{x_i}[\ell]$= bit ℓ of $\boldsymbol{x_i}$
  Observe: every user can compute her row locally

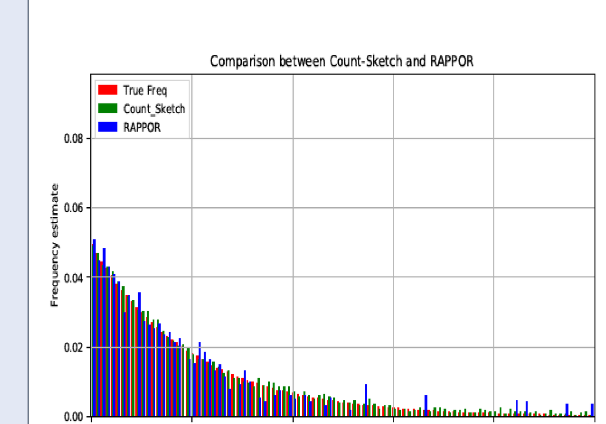
$\boldsymbol{x}^*$ is "heavy", hence $(\boldsymbol{t}^*, \boldsymbol{x}^*[\ell])$ appears $> \boldsymbol{2\tau}$ in $\boldsymbol{S}_\ell$ for every ℓ
No collisions, hence $(\boldsymbol{t}^*, \boldsymbol{1} - \boldsymbol{x}^*[\ell])$ appears $\boldsymbol{0}$ times in $\boldsymbol{S}_\ell$
$\Longrightarrow$ Can identify every bit ℓ of $\boldsymbol{x}^*$ by querying $\boldsymbol{\mathcal{O}}(\boldsymbol{S}_\ell)$ on $(\boldsymbol{t}^*, \boldsymbol{0})$ and $(\boldsymbol{t}^*, \boldsymbol{1})$

The Protocol: For every $\boldsymbol{t} \in [\boldsymbol{T}]$ construct $\hat{\boldsymbol{x}}^{(\boldsymbol{t})}$ as follows:
$\forall \ell \in [\log|\boldsymbol{X}|]$ query $\boldsymbol{\mathcal{O}}(\boldsymbol{S}_\ell)$ on $(\boldsymbol{t}, \boldsymbol{0})$ and $(\boldsymbol{t}, \boldsymbol{1})$ and set $\hat{\boldsymbol{x}}^{(\boldsymbol{t})}[\ell] \leftarrow$ **argmax**

- By purple, $\hat{\boldsymbol{x}}^{(\boldsymbol{t}^*)} = \boldsymbol{x}^*$ is identified

Empirical evaluation with TreeHist

Comparison to RAPPOR project:
ϵ = 1.09, n= 1 mi, domain = 100



Heavy hitters for NLTK data set:
ϵ = 2, n= 10 mi, # of unique words = 25991, wordlength = 6
Threshold= $15\sqrt{n}$

True positives: 22	True negatives: $\approx 3 \times 10^8$
False positives: 60	False negatives: 3