UNIVERSITÀ DEGLI STUDI DI PADOVA

DIFFERENTIAL PRIVACY: PART III

PRIVACY PRESERVING INFORMATION ACCESS

PhD in Information Engineering
A.Y. 2025/2026

GUGLIELMO FAGGIOLI
Intelligent Interactive Information Access (IIIA) Hub
Department of Information Engineering
University of Padua

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CENTRAL VS LOCAL DP

Most of the techniques we have seen until now are examples of **central DP**: we assume to have a centralised entity (e.g., a server) that holds all the data and needs to privatize it to release it.

This has some limitations: the user is expected to release their information to the central server.

Local DP: on the user’s side, the user applies some DP approach on their data and releases it to the central server. The server will then aggregate the results.

IS THERE SOMEONE THAT ACTUALLY USES DP?!!!

GOOGLE: RAPPOR

RAPPOR is vaguely based on the coin toss algorithm and is used by google to collect stats on chrome and possibly on the machines it works on.

For example, it allows to get information about the most popular homepage.

GOOGLE: RAPPOR

RAPPOR is based on cohorts of users: the users are first clustered (randomly) in groups of users. This is not strictly necessary, and you can apply RAPPOR with a single cohort, but the results will be less accurate.

The transmission of a value v to the server is based on three phases:

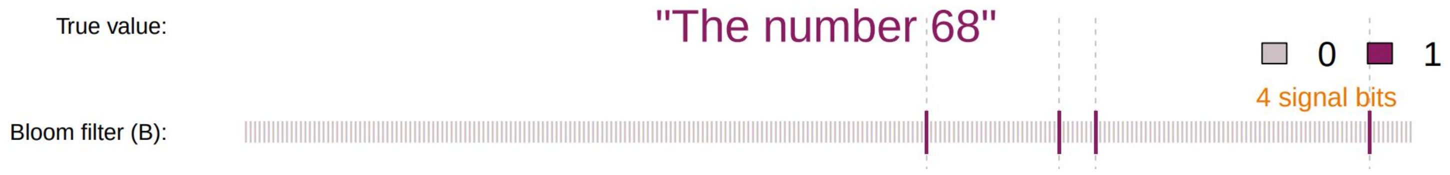
1) encode the value v in a vector B via a bloom filter
2) randomize (and **memoize**) B in a vector B’
3) randomize B’ in a response S and send it to the server

GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B’
3) randomize B’ in a response S and send it to the server

Using h hash functions encode a value in a vector of size k



GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B’
3) randomize B’ in a response S and send it to the server

$$B'_i = \begin{cases} 1, & \text{with probability } \frac{1}{2}f \\ 0, & \text{with probability } \frac{1}{2}f \\ B_i, & \text{with probability } 1-f \end{cases}$$

This is almost exactly the coin toss!

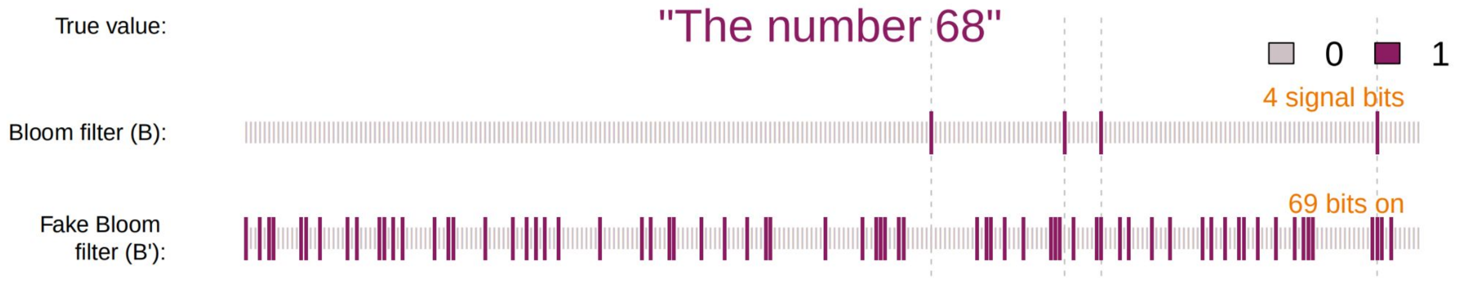
f is a parameter that can be user-tuned.

B’ is **memoized** and used every time the value *v* needs to be returned. This prevents the **averaging attack**.

GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B'
3) randomize B' in a response S and send it to the server



GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B’
3) randomize B’ in a response S and send it to the server

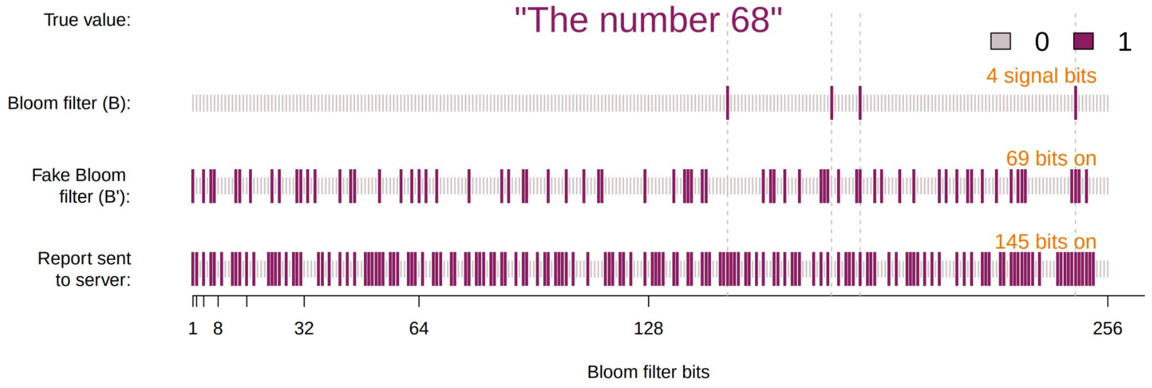
$$P(S_i = 1) = \begin{cases} q, & \text{if } B'_i = 1 \\ p, & \text{if } B'_i = 0 \end{cases}$$

q and p are again parameters that regulate the tradeoff between the accuracy and the privacy.

GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B’
3) randomize B’ in a response S and send it to the server



GOOGLE: RAPPOR

The transmission of a value v to the server is based on three phases:

1) encode the value v in a vector B via a bloom filter
2) randomize (and memoize) B in a vector B’
3) randomize B’ in a response S and send it to the server

RAPPOR is (proven to be) ε-DP, with:

$$\varepsilon = 2h \log \left(\frac{1 - \frac{1}{2} f}{\frac{1}{2} f} \right)$$

GOOGLE: RAPPOR

It is impossible to determine the original true value of a bit from a report, but it is possible to estimate the proportion of bits truly set to 1 (similar to what we did with the true YES for the coin toss).

$$t_{ij} = \frac{c_{ij} - \left(p + \frac{1}{2}fq - \frac{1}{2}fp\right)N_j}{(1-f)(q-p)}$$

number of times i is set to 1 in cohort j

number of reports received by cohort j

Rappor divides users into cohorts with different hash functions (to reduce collisions). Nevertheless, the number of cohorts might be even 1.

GOOGLE: RAPPOR

Then, using such frequencies, you can compute linear model (as in the coin toss algorithm) that “tells” you the (perturbed) frequency of a set of candidate strings.

For example, given a set of “most likely” homepages for chrome (e.g., “google.com”, “bing.com” etc): they compute how popular are these homepages.

Given the name of a known malicious binary, they “discovered” that the malicious binary is the 8th most common process running on Windows machine!

APPLE: SEQUENCE FRAGMENT PUZZLE

RAPPOR has the **limitation of requiring a list of “candidate strings”**: it does not allow to discover new words (unless you guess them!)

The Apple DP strategy allows to overcome this limitation.

APPLE: COUNT MEAN SKETCH & SEQUENCE FRAGMENT PUZZLE

www.example.com

APPLE: COUNT MEAN SKETCH (CMS)

1) Randomly select a hash function among a set of hash functions $\{h_1, h_2, \ldots, h_n\}$. Let say you have sampled the third hash function h_3

2) hash the string: h_3(www.example.com) = 31

3) define a vector B such that all the elements are equal to 0 except 31

4) flip elements in B with probability $1/(e^{\varepsilon/2}+1)$

5) send [B, 3] to the server, where 3 is the index of the hash function

APPLE: COUNT MEAN SKETCH (CMS)

Server side.

1) construct a matrix M with size h x m, where h is the number of hash functions considered and m is the dimension of B

2) add B to the i-th row of M, where i is the index of the sampled hash function

3) Debias M: you know that $1/(e^{\varepsilon/2}+1)$ of the ones you have received where in fact zeros, and vice-versa (as in the coin-toss)!

4) to compute the frequency of www.example.com, average, over all i M[i, h_i(www.example.com)]

APPLE: COUNT MEAN SKETCH (CMS)

Server side.

1) construct a matrix M with size h x m, where h is the number of hash functions considered and m is the dimension of B

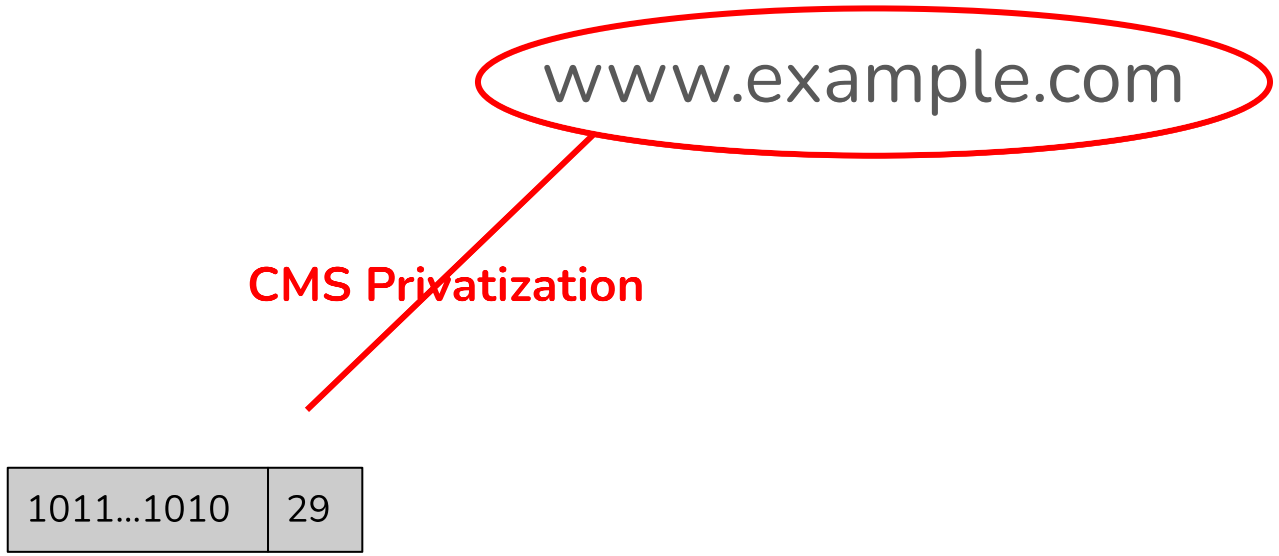
Cool, but … we still need to know the string we are looking for

2) add B to the i-th row of M, where i is the index of the sampled hash function

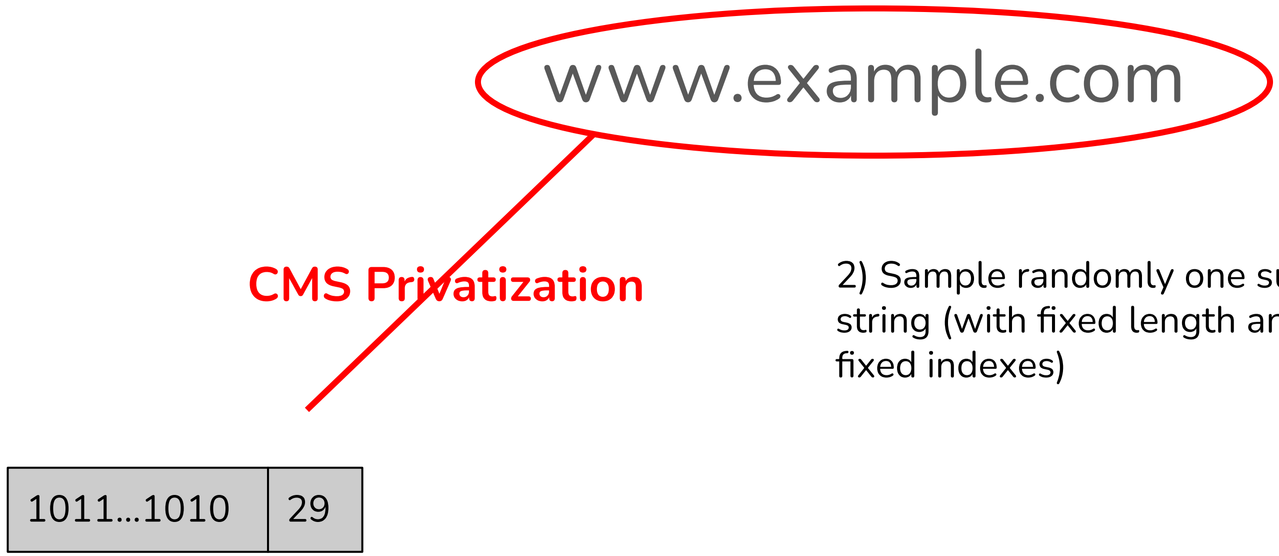
3) Debias M: you know that $1/(e^{\varepsilon/2}+1)$ of the ones you have received where in fact zeros (as in the coin-toss)!

4) to compute the frequency of www.example.com, average, over all i $M[i, h_i(\text{www.example.com})]$

APPLE: SEQUENCE FRAGMENT PUZZLE

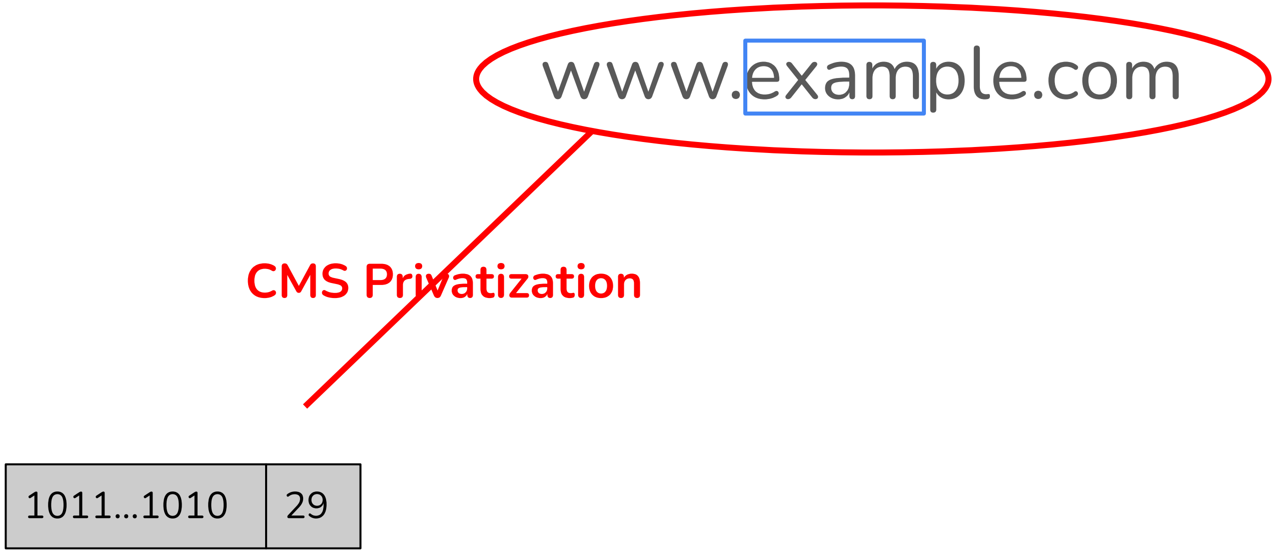


APPLE: SEQUENCE FRAGMENT PUZZLE



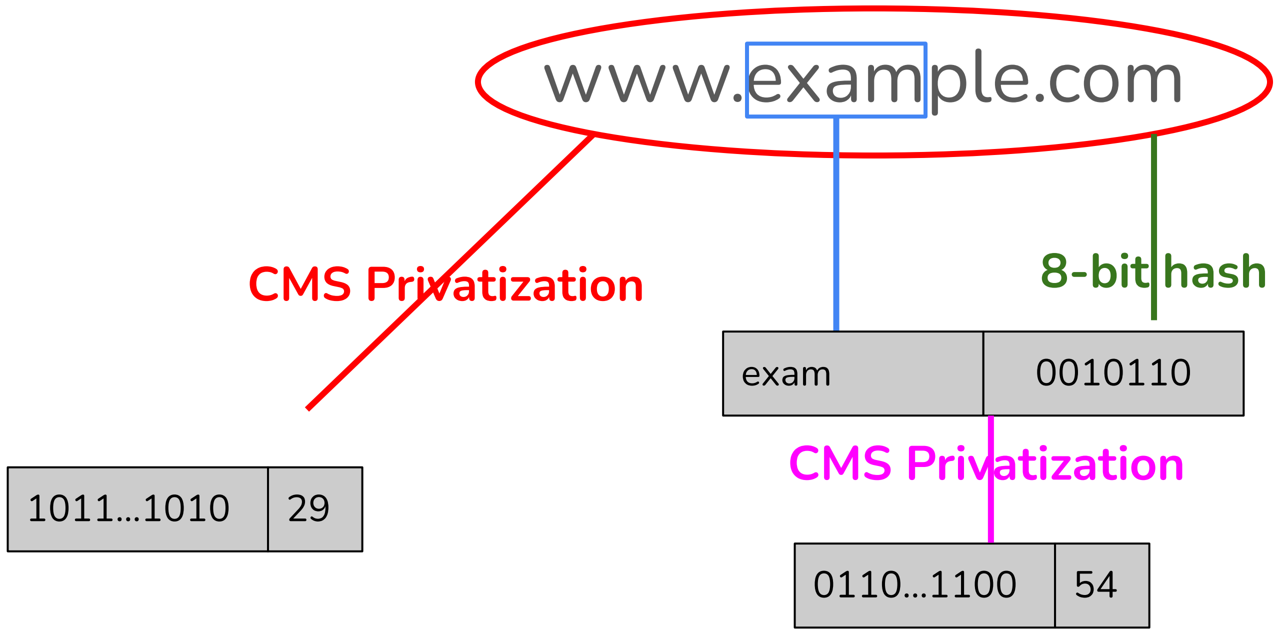
2) Sample randomly one substring of the string (with fixed length and starting from fixed indexes)

APPLE: SEQUENCE FRAGMENT PUZZLE

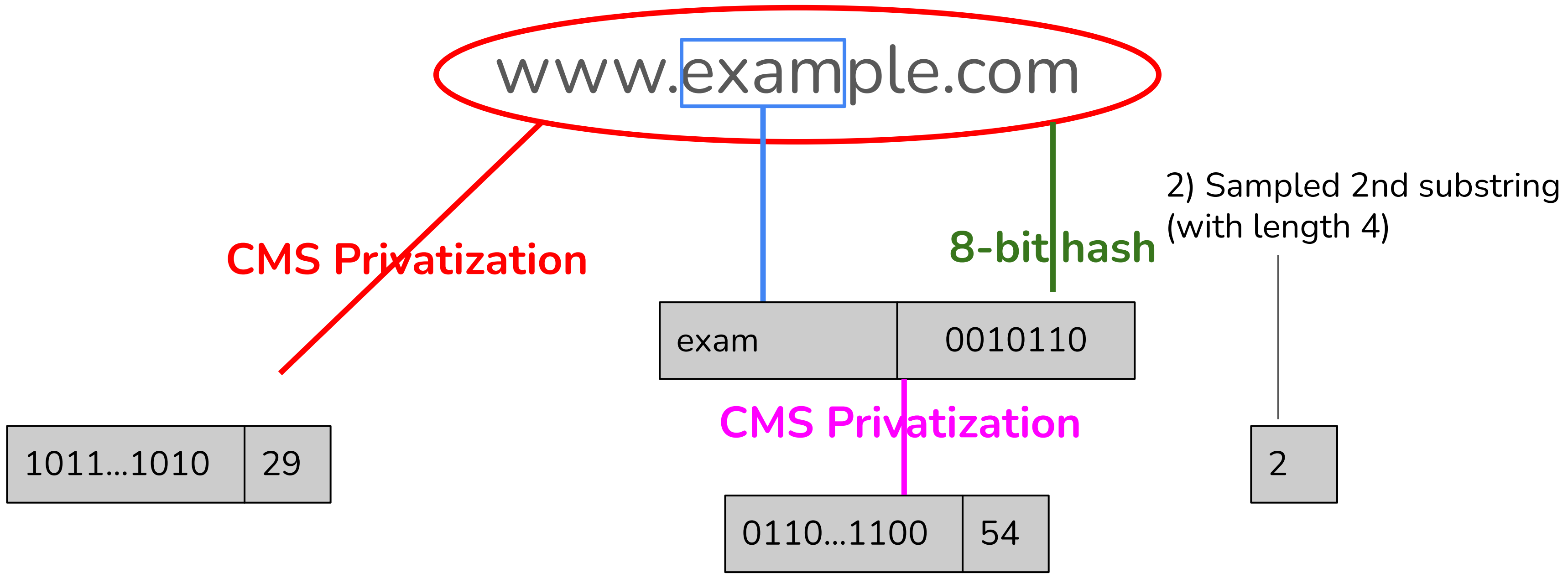


2) Sampled 2nd substring (with length 4)

APPLE: SEQUENCE FRAGMENT PUZZLE



APPLE: SEQUENCE FRAGMENT PUZZLE



Send everything to the server

APPLE: SEQUENCE FRAGMENT PUZZLE

Easy to enumerate all the possible 4-characters substrings.

For all possible 4-characters substrings, we concatenate them with the small hash and count the most frequent substrings starting at each considered index.

We can then combine most frequent substrings (that have the same small hashes) to discover new previously unseen strings.

MICROSOFT: LOW COMMUNICATION LDP

Assume you want to discover the mean time spent by **all** users on a certain application.

You can ask each user to send you the time they spend on that application and average it.

MICROSOFT: LOW COMMUNICATION LOCAL DP (LDP)

Assume you want to discover the mean time spent by **all** users on a certain application.

You can ask each user to send you the time they spend on that application and average it.

Obviously, you cannot.

MICROSOFT: LOW COMMUNICATION LDP

Assume you want to discover the mean time spent by **all** users on a certain application.

You can ask each user to send you the time they spend on that application with a certain Laplace noise and average it.

$\mathrm{Lap}(\Delta f/\varepsilon)$

MICROSOFT: LOW COMMUNICATION LDP

Assume you want to discover the mean time spent by **all** users on a certain application.

You can ask each user to send you the time they spend on that application with a certain Laplace noise and average it.

What is the sensitivity in this case? (how much the function changes if I do not participate on the counting?)

MICROSOFT: LOW COMMUNICATION LDP

Assume you want to discover the mean time spent by **all** users on a certain application.

You can ask each user to send you the time they spend on that application with a certain Laplace noise and average it.

What is the sensitivity in this case? the maximum time I can spend on the application - m.

MICROSOFT: LOW COMMUNICATION LDP

how many bits do I need to send this information?

assuming m is the maximum value that I can send, I will need $\log_2(m)$ bits to encode this information.

Not that small… can we do better?

MICROSOFT: LOW COMMUNICATION LDP

called X_i the time you spend on a certain application, with X_i in [0, m] send Y_i , a single bit, such that:

$$P(Y_i = 1) = \frac{1}{e^{\varepsilon}+1} + \frac{X_i}{m} \cdot \frac{e^{\varepsilon}-1}{e^{\varepsilon}+1}$$

Then, to compute the average time spent on the application, use the following formula:

$$\hat{\mu} = \frac{m}{n} \sum_i \frac{Y_i \cdot (e^{\varepsilon}+1) - 1}{e^{\varepsilon}-1}$$

MICROSOFT: LOW COMMUNICATION LDP

does it work?

$$
\begin{aligned}
E[\hat{\mu}] &= \frac{m}{n}\sum_i \frac{E[Y_i]\cdot(e^\varepsilon+1)-1}{e^\varepsilon-1} \\
&= \frac{m}{n}\sum_i \frac{\left(\frac{1}{e^\varepsilon+1}+\frac{X_i}{m}\cdot\frac{e^\varepsilon-1}{e^\varepsilon+1}\right)\cdot(e^\varepsilon+1)-1}{e^\varepsilon-1} \\
&= \frac{1}{n}\sum_i X_i
\end{aligned}
$$

A single (almost) random bit and still you are able to extract useful information... isn’t it powerful?

US CENSUS BUREAU

US Census Bureau use a composition of two differential privacy approaches: one used to protect persons files (ε=17.14) and one to protect housing data (ε=2.47).

Thus, their mechanism is 19.61-DP.

US CENSUS BUREAU

The census bureau applies some noise to the population counting.

They leave invariant the total number of people at state level, but hey perturb all the statistics at lower level (counties, cities, census blocks).

Their approach tend to introduce much variance to lower levels (remember the composition problem?).

US CENSUS BUREAU

Interestingly this has prompted a lively discussion about the effects of the differential privacy in the national congress.

For example the national congress of american indians notes that "*The implementation of differential privacy could introduce substantial amounts of noise into statistics for small populations living in remote areas, potentially diminishing the quality of statistics about tribal nations*."

"*How much should usability be compromised to protect confidentiality and vice versa, since both are legal requirements?*"

US CENSUS BUREAU

More details here:

https://www.ncsl.org/research/redistricting/differential-privacy-for-census-data-explained.aspx