Differential privacy without a central database

Boston Differential Privacy Summer School, 6-10 June 2022

Uri Stemmer

About this course

- The local model ✓
- The shuffle model ✓
- Streaming/online settings
- Differential privacy as a tool

Streaming/online settings

Today's Outline

1. Private streaming algorithms

2. Privacy under continual observation

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Example: Can you count the number of distinct characters?

What is Streaming?

Alon, Matias, Szegedy 96

Correct answer: 9

What is Streaming?

Alon, Matias, Szegedy 96

Correct answer: 9

- A stream of length $\boldsymbol{n}$ over domain $\boldsymbol{X}$ is a sequence of updates $(\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$ where $\boldsymbol{x_i} \in \boldsymbol{X}$
- Let $\boldsymbol{g}: \boldsymbol{X}^* \to \boldsymbol{R}$ be a function
- At every time $\boldsymbol{i} \in [\boldsymbol{n}]$ we obtain $\boldsymbol{x_i}$
- At the end of the stream we need to output $\boldsymbol{z} \approx \boldsymbol{g}(\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
- **Requirement: small space**

What is Streaming?

Alon, Matias, Szegedy 96

Correct answer: 9

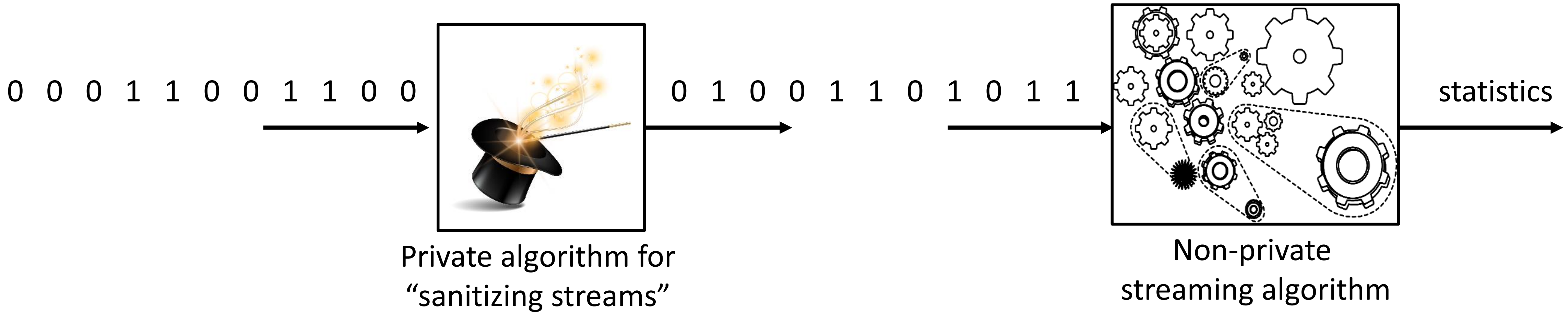
- A stream of length $\boldsymbol{n}$ over domain $\boldsymbol{X}$ is a sequence of updates $(\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$ where $\boldsymbol{x_i} \in \boldsymbol{X}$
- Let $\boldsymbol{g}: \boldsymbol{X}^* \to \boldsymbol{R}$ be a function
- At every time $\boldsymbol{i} \in [\boldsymbol{n}]$ we obtain $\boldsymbol{x_i}$
- At the end of the stream we need to output $\boldsymbol{z} \approx \boldsymbol{g}(\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
- **Requirement: small space**

- What does it mean for a streaming algorithm to be DP?

  - A streaming algorithm $\mathcal{A}$ is (ε, δ)-DP if for any two neighboring streams $\vec{x} = (x_1, \dots, x_n)$ and $\vec{x}' = (x_1', \dots, x_n')$ that differ on one update we have that $\mathcal{A}(\vec{x}) \approx_{(\varepsilon,\delta)} \mathcal{A}(x')$

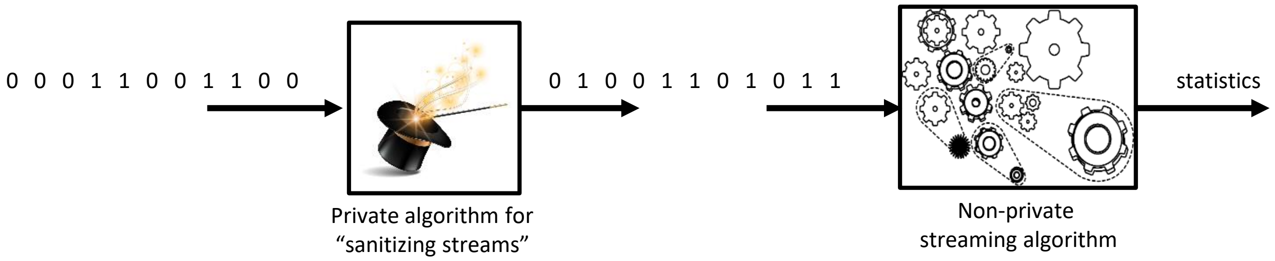
How can we design private streaming algorithms?

Idea 1: Synthetic streams



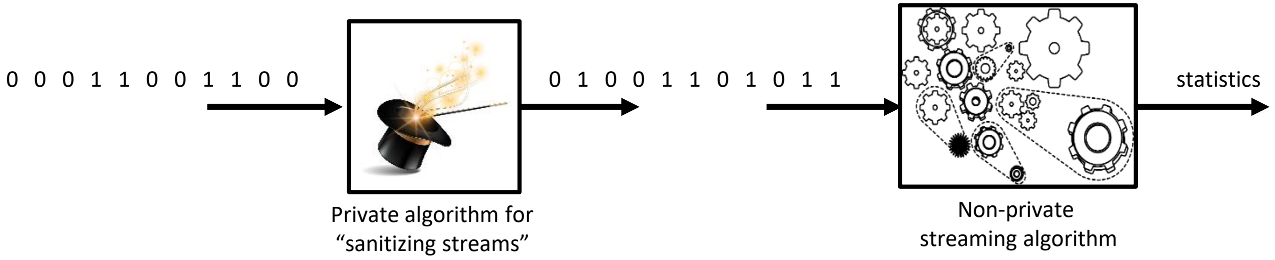
How can we design private streaming algorithms?

Idea 1: Synthetic streams



How can we design private streaming algorithms?

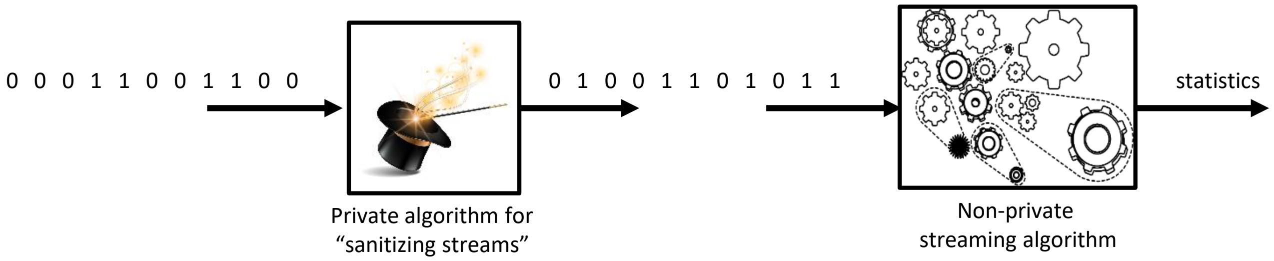
Idea 1: Synthetic streams



- For a predefined family of queries $\boldsymbol{Q}$, a “streaming sanitizer” is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{\mathbf{1}}{|\boldsymbol{S}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\boldsymbol{x}) \approx \frac{\mathbf{1}}{|\widehat{\boldsymbol{S}}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\widehat{\boldsymbol{x}})$
- Recall the definition of standard “offline sanitizers” which is the same except that the input and outputs are specified all at once…

How can we design private streaming algorithms?

Idea 1: Synthetic streams



- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{\mathbf{1}}{|\boldsymbol{S}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\boldsymbol{x}) \approx \frac{\mathbf{1}}{|\widehat{\boldsymbol{S}}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\widehat{\boldsymbol{x}})$

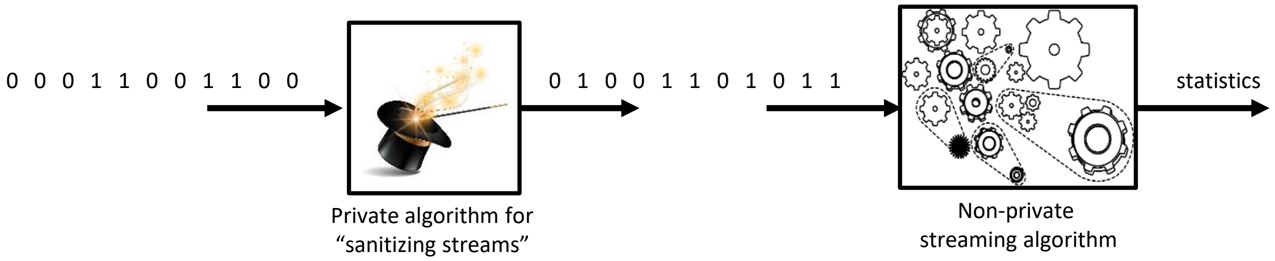
- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…

- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$

- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)

How can we design private streaming algorithms?

Idea 1: Synthetic streams



- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{1}{|S|}\sum_{x \in S} q(x) \approx \frac{1}{|\hat{S}|}\sum_{x \in S} q(\hat{x})$

- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…

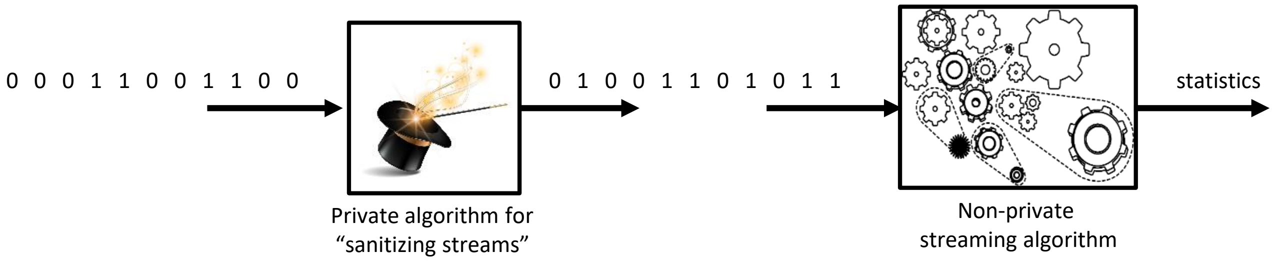
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$

- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)

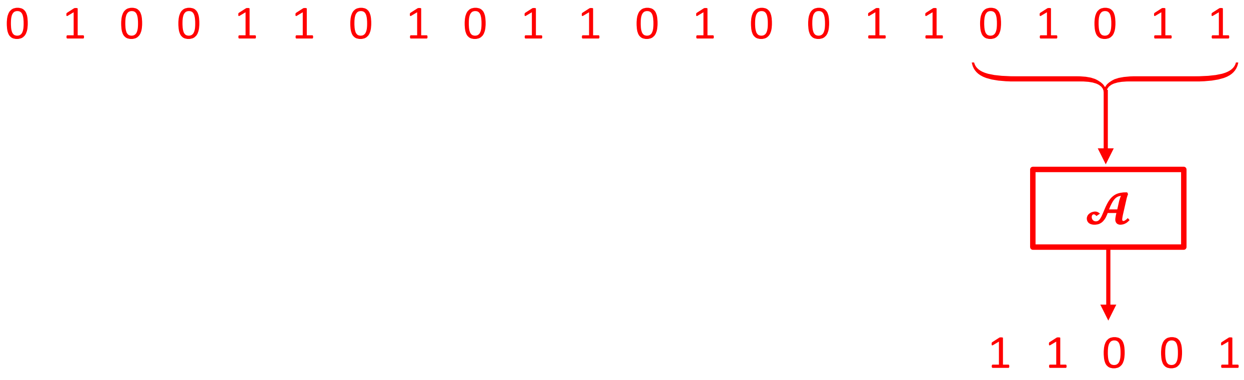
0 1 0 0 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 0 1 1

How can we design private streaming algorithms?

Idea 1: Synthetic streams

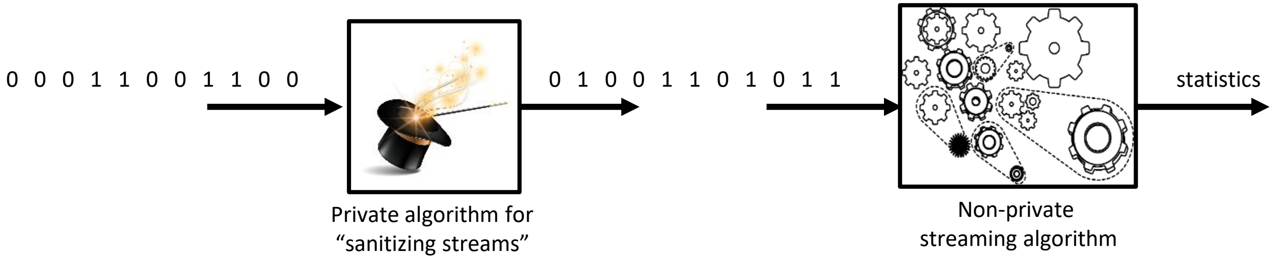


- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{\mathbf{1}}{|\boldsymbol{S}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\boldsymbol{x}) \approx \frac{\mathbf{1}}{|\widehat{\boldsymbol{S}}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\widehat{\boldsymbol{x}})$
- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$
- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)

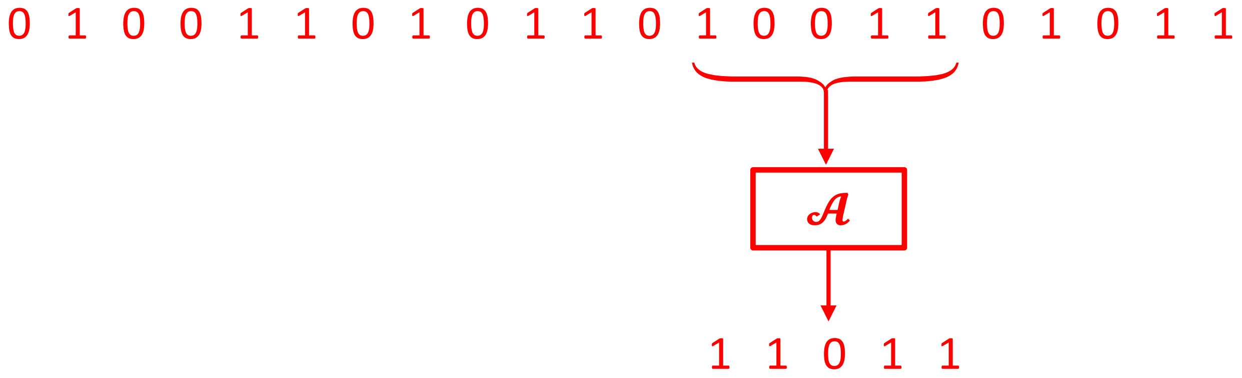


How can we design private streaming algorithms?

Idea 1: Synthetic streams

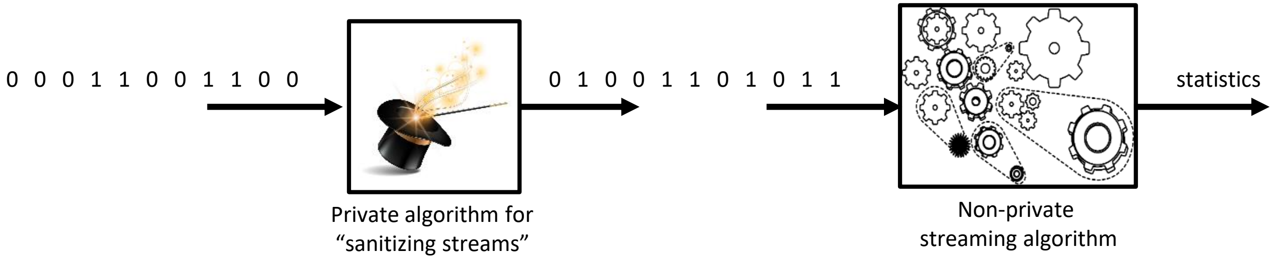


- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{1}{|S|}\sum_{x\in S} \boldsymbol{q}(\boldsymbol{x}) \approx \frac{1}{|\hat{S}|}\sum_{x\in S} \boldsymbol{q}(\widehat{\boldsymbol{x}})$
- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$
- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)



How can we design private streaming algorithms?

Idea 1: Synthetic streams

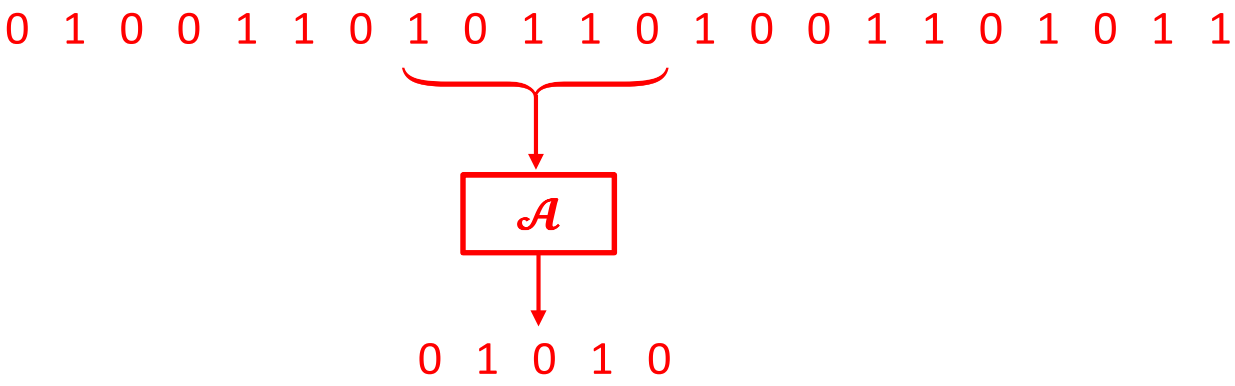


- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{\mathbf{1}}{|\boldsymbol{S}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\boldsymbol{x}) \approx \frac{\mathbf{1}}{|\widehat{\boldsymbol{S}}|}\sum_{\boldsymbol{x}\in\boldsymbol{S}} \boldsymbol{q}(\widehat{\boldsymbol{x}})$

- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…

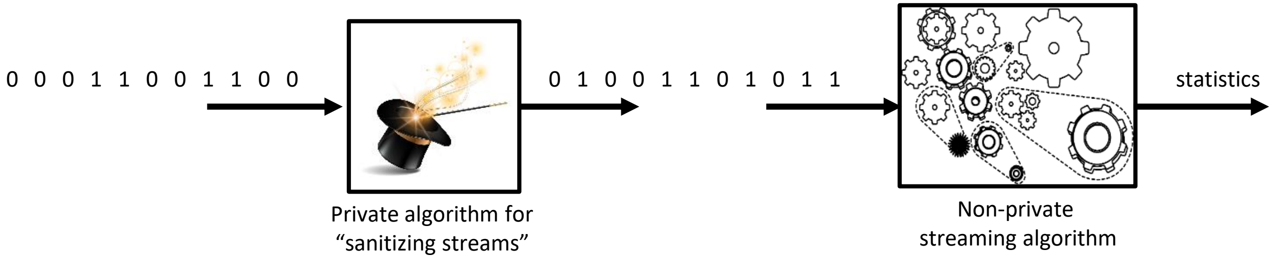
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$

- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)

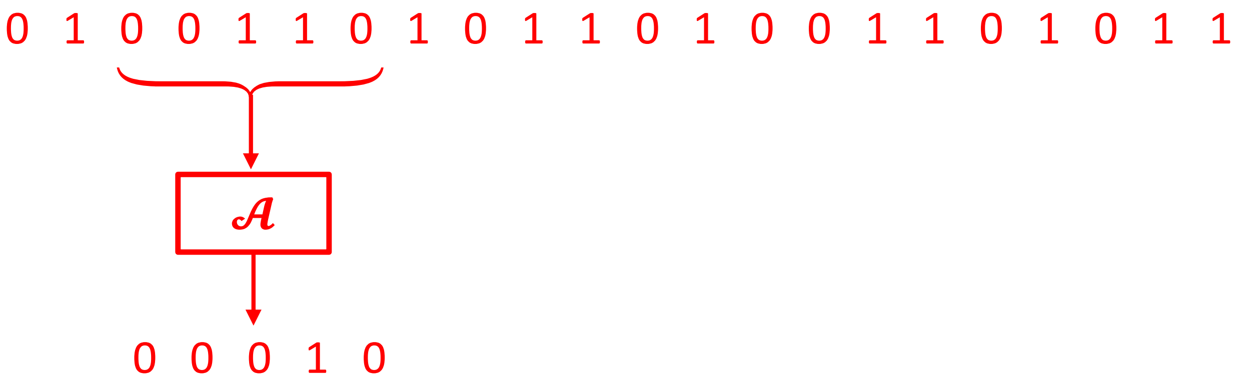


How can we design private streaming algorithms?

Idea 1: Synthetic streams

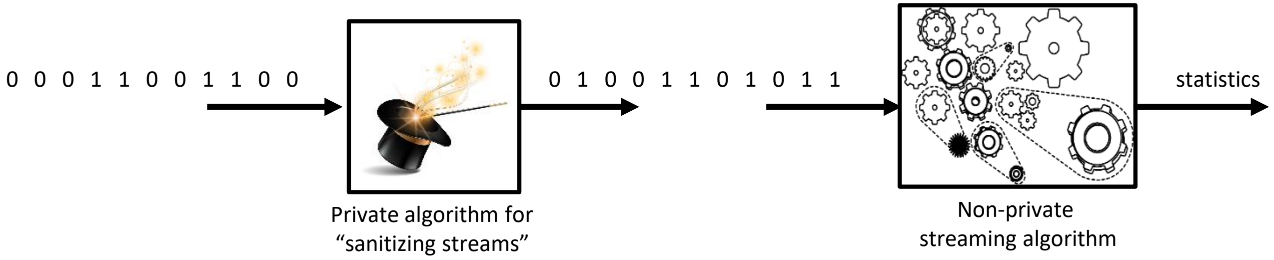


- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{1}{|S|}\sum_{x\in S} q(x) \approx \frac{1}{|\hat{S}|}\sum_{x\in S} q(\hat{x})$
- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\mathcal{A}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$
- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\mathcal{A}(\boldsymbol{D})$
  3) goto step (1)



How can we design private streaming algorithms?

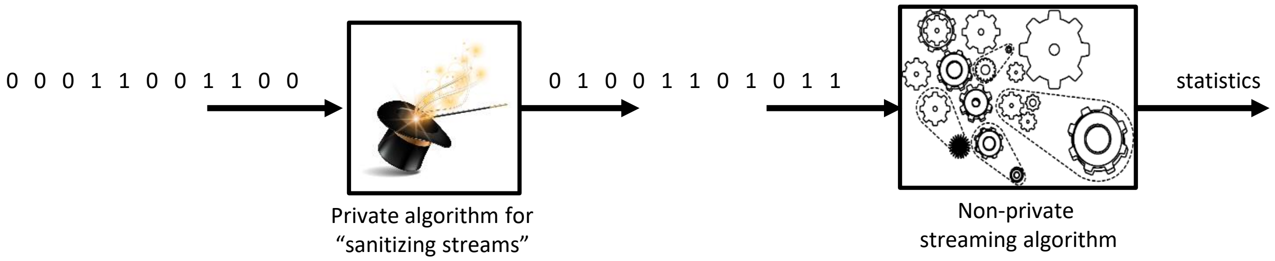
Idea 1: Synthetic streams



- For a predefined family of queries $\boldsymbol{Q}$, a “streaming sanitizer” is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{1}{|S|}\sum_{x\in S} q(x) \approx \frac{1}{|\widehat{S}|}\sum_{x\in S} q(\widehat{x})$
- Recall the definition of standard “offline sanitizers” which is the same except that the input and outputs are specified all at once…
- Standard “offline sanitizers” can be transformed into “streaming sanitizers”: Suppose we have an “offline sanitizer” $\boldsymbol{\mathcal{A}}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$
- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\boldsymbol{\mathcal{A}}(\boldsymbol{D})$
  3) goto step (1)

How can we design private streaming algorithms?

Idea 1: Synthetic streams



- For a predefined family of queries $\boldsymbol{Q}$, a "streaming sanitizer" is a DP algorithm whose input is a stream $\boldsymbol{S}$ and its output is a stream $\widehat{\boldsymbol{S}}$ such that for every $\boldsymbol{q} \in \boldsymbol{Q}$ we have $\frac{1}{|S|}\sum_{x \in S} q(x) \approx \frac{1}{|\hat{S}|}\sum_{x \in S} q(\hat{x})$
- Recall the definition of standard "offline sanitizers" which is the same except that the input and outputs are specified all at once…
- Standard "offline sanitizers" can be transformed into "streaming sanitizers": Suppose we have an "offline sanitizer" $\mathcal{A}$ for $\boldsymbol{Q}$ that operates on datasets of size $\boldsymbol{m}$
- Construct a streaming sanitizer as follows:
  1) Let $\boldsymbol{D}$ denote the next $\boldsymbol{m}$ items
  2) Output $\mathcal{A}(\boldsymbol{D})$
  3) goto step (1)

Privacy analysis:

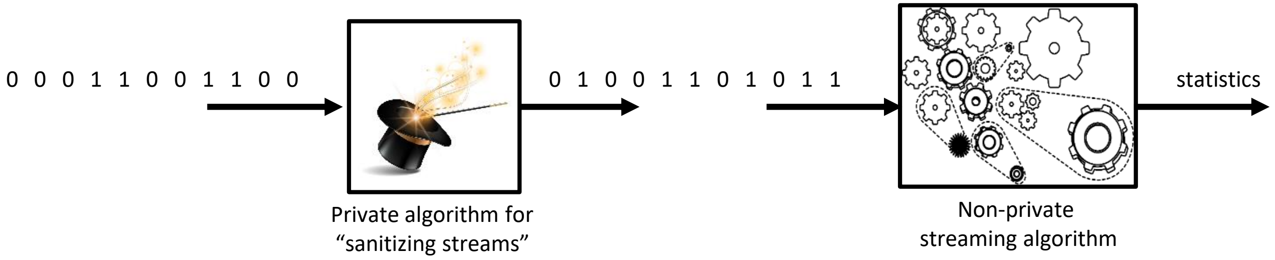
- We apply $\mathcal{A}$ on disjoint portions of the input
- So no need for composition and privacy follows from $\mathcal{A}$

Utility analysis:

- Since we aim for <u>relative</u> error, the error do not accumulate

How can we design private streaming algorithms?

Idea 1: Synthetic streams

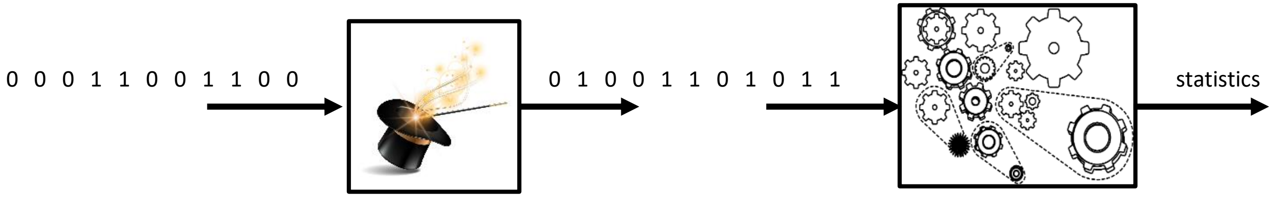


- The point here is that our space does not depend directly on the length of the stream n
- Our space equals to the space of the sanitizer $\mathcal{A}$, which is independent of n, and the space of the non-private streaming algorithm

- Since we aim for relative error, the error do not accumulate

How can we design private streaming algorithms?

Idea 1: Synthetic streams



- F
  i
- F
  s
- S
  s
- C

- The point here is that our space does not depend directly on the length of the stream n
- Our space equals to the space of the sanitizer $\mathcal{A}$, which is independent of n, and the space of the non-private streaming algorithm

Example where this is useful: Quantile estimation

- Items in the stream are numbers $x_1, x_2, \ldots, x_n \in [0,1]$
- The goal is, at the end of the stream, to get approximations for all quantiles of the data
- E.g., at the end of the stream we want to learn 9 numbers $y_1, y_2, \ldots, y_9 \in [0,1]$ such that for every ℓ we have $|\{i: y_\ell \leq x_i \leq y_{\ell+1}\}| \approx \frac{n}{10}$

(this works well because we have very efficient “offline sanitizers” for this problem)

- Since we aim for relative error, the error do not accumulate

The Counter Problem

Morris 78

Updates are bits and we want to estimate their sum

- Simple solution: Store the sum in memory using $\log n$ bits
  - Can we maintain a counter using smaller space?

The Counter Problem

Morris 78

Updates are bits and we want to estimate their sum

- Simple solution: Store the sum in memory using $\log n$ bits
    - Can we maintain a counter using smaller space?

Step 1:

— Initialize estimate $\widehat{\boldsymbol{C}} = \mathbf{0}$
— Given an update $\boldsymbol{x_i} = \mathbf{1}$ flip a coin and increment $\widehat{\boldsymbol{C}}$ only if coin is heads

- We expect that $\widehat{\boldsymbol{C}} \approx \boldsymbol{C}/\mathbf{2}$ where $\boldsymbol{C}$ is the true value
- The good: We still know $\boldsymbol{C}$ (approximately) while storing only a smaller number
- The bad: We saved only 1 bit (and also this has large variance, but let's ignore it…)

The Counter Problem

Morris 78

Updates are bits and we want to estimate their sum

- Simple solution: Store the sum in memory using $\log n$ bits
  - Can we maintain a counter using smaller space?

Step 1:

— Initialize estimate $\widehat{C} = 0$
— Given an update $x_i = 1$ flip a coin and increment $\widehat{C}$ only if coin is heads

- We expect that $\widehat{C} \approx C/2$ where C is the true value
- The good: We still know C (approximately) while storing only a smaller number
- The bad: We saved only 1 bit (and also this has large variance, but let's ignore it…)

Step 2:

— Given an update $x_i = 1$ flip $\widehat{C}$ **coins** and increment $\widehat{C}$ only if **all** coin are heads

- Can show that in expectation $\widehat{C} \approx \log C$
- Thus if C takes $\log n$ then $\widehat{C}$ takes $\approx \log \log n$ bits
- So we gained exponentially in storage! (again, let's ignore the variance…)

The Counter Problem

Morris 78

Updates are bits and we want to estimate their sum

- Simple solution: Store the sum in memory using $\log n$ bits
  - Can we maintain a counter using smaller space?

Step 1:

— Initialize estimate $\hat{C} = 0$
— Given an update $x_i = 1$ flip a coin and increment $\hat{C}$ only if coin is heads

- We expect that $\hat{C} \approx C/2$ where C is the true value
- The good: We still know C (approximately) while storing only a smaller number
- The bad: We saved only 1 bit (and also this has large variance, but let's ignore it…)

Step 2:

— Given an update $x_i = 1$ flip $\hat{C}$ **coins** and increment $\hat{C}$ only if **all** coin are heads

- Can show that in expectation $\hat{C} \approx \log C$
- Thus if C takes $\log n$ then $\hat{C}$ takes $\approx \log \log n$ bits
- So we gained exponentially in storage! (again, let's ignore the variance…)

Observe: The outcome distribution of the algorithm depends only on C

A Private Algorithm for the Counter Problem

[Dwork, Naor, Pitassi, Rothblum, Yekhanin]

- We can design a private variant as follows (informal):

— Sample $\boldsymbol{Y} \sim \mathbf{Lap}\left(\frac{\mathbf{1}}{\boldsymbol{\varepsilon}}\right)$

— Run Morris' counter on a modified stream:

- If $\boldsymbol{Y} < \mathbf{0}$ then ignore the first $|\boldsymbol{Y}|$ ones in the stream
- If $\boldsymbol{Y} \geq \mathbf{0}$ then add $\boldsymbol{Y}$ ones before the stream begins

— The outcome distribution if a function of $(\boldsymbol{C} + \boldsymbol{Y})$, satisfying privacy by post-processing

A Private Algorithm for the Counter Problem

[Dwork, Naor, Pitassi, Rothblum, Yekhanin]

- We can design a private variant as follows (informal):

— Sample $\boldsymbol{Y} \sim \mathbf{Lap}\left(\frac{1}{\varepsilon}\right)$
— Run Morris' counter on a modified stream:
  - If $\boldsymbol{Y} < \mathbf{0}$ then ignore the first $|\boldsymbol{Y}|$ ones in the stream
  - If $\boldsymbol{Y} \geq \mathbf{0}$ then add $\boldsymbol{Y}$ ones before the stream begins

— The outcome distribution if a function of $(\boldsymbol{C} + \boldsymbol{Y})$, satisfying privacy by post-processing

- This idea is useful for other streaming problems

➢ Example in the context of the counter problem: Counting the number of people who viewed my YouTube video

Streaming/online settings

Today's Outline

1. Private streaming algorithms
2. Privacy under continual observation

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Modified problem – Counter with continual reports:

- On every time $\boldsymbol{t} \in [\boldsymbol{n}]$
  - We get a bit $\boldsymbol{x_t} \in \{\mathbf{0}, \mathbf{1}\}$
  - Need to respond with an approximation $\hat{\boldsymbol{c}}_{\boldsymbol{i}}$ for $\boldsymbol{c_i} = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{t}} \boldsymbol{x_i}$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Modified problem – Counter with continual reports:

- On every time $\boldsymbol{t} \in [\boldsymbol{n}]$
  - We get a bit $\boldsymbol{x_t} \in \{\mathbf{0}, \mathbf{1}\}$
  - Need to respond with an approximation $\hat{\boldsymbol{c}}_{\boldsymbol{i}}$ for $\boldsymbol{c_i} = \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{t}} \boldsymbol{x_i}$

➢ Algorithm $\mathcal{A}$ is (ε, δ)-DP for this problem if for any two neighboring streams $\vec{x} = (x_1, \dots, x_n)$ and $\vec{x}' = (x_1', \dots, x_n')$ that differ on one update we have that $\mathcal{A}(\vec{u}) \approx_{(\varepsilon,\delta)} \mathcal{A}(\vec{u}')$

Remarks:

- Observe that now $\mathcal{A}(\vec{x})$ is a vector of length m
- This problem is interesting regardless of space, so let's forget about space from now on
- Sanity check: Is the previous algorithm private w.r.t. this definition?

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Naïve attempts at solving the problem:

1) "LDP style": Every time $\boldsymbol{t} \in [\boldsymbol{n}]$ we release $\widehat{\boldsymbol{x}}_{\boldsymbol{t}} = \boldsymbol{x}_{\boldsymbol{t}} + \mathbf{Lap}\left(\frac{\mathbf{1}}{\boldsymbol{\varepsilon}}\right)$

   —This would maintain privacy, but sum of $\boldsymbol{n}$ noises accumulates to $\approx \sqrt{\boldsymbol{n}}/\boldsymbol{\varepsilon}$

2) Using composition: Every time $\boldsymbol{t} \in [\boldsymbol{n}]$ we release $\hat{\boldsymbol{c}}_{\boldsymbol{t}} = \left(\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{t}} \boldsymbol{x}_{\boldsymbol{t}}\right) + \mathbf{Lap}(\boldsymbol{b})$

   —We would need $\boldsymbol{b} \approx \sqrt{\boldsymbol{n}}/\boldsymbol{\varepsilon}$ due to composition

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Naïve attempts at solving the problem:

1) "LDP style": Every time $t \in [n]$ we release $\hat{x}_t = x_t + \mathrm{Lap}\left(\frac{1}{\varepsilon}\right)$

   —This would maintain privacy, but sum of n noises accumulates to $\approx \sqrt{n}/\varepsilon$

2) Using composition: Every time $t \in [n]$ we release $\hat{c}_t = \left(\sum_{i=1}^{t} x_t\right) + \mathrm{Lap}(b)$

   —We would need $b \approx \sqrt{n}/\varepsilon$ due to composition

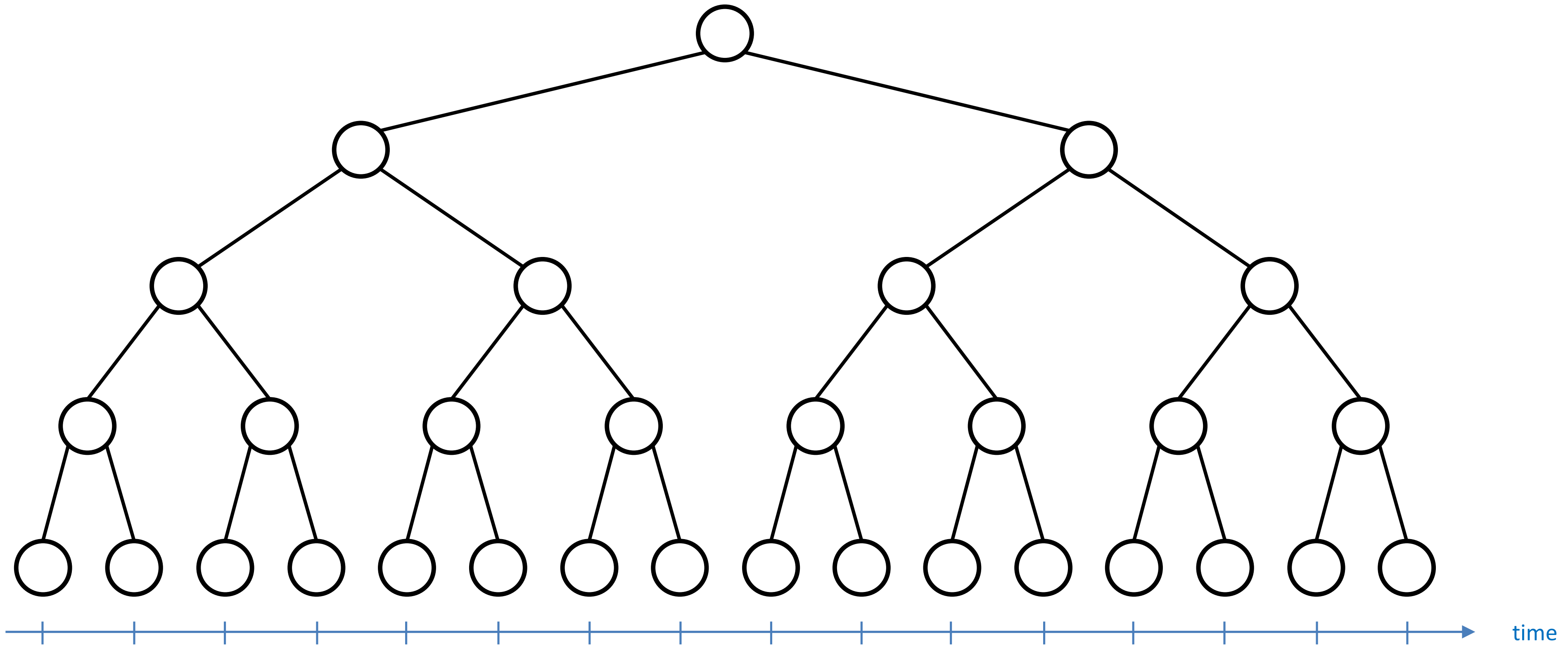
How can we do better?

- Observe: in solution (1) every user affects only one computation, so no need for composition, but the noises accumulate. In solution (2) we do not accumulate noises, but each one must be big to account for composition over n computations
- We want something in between

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

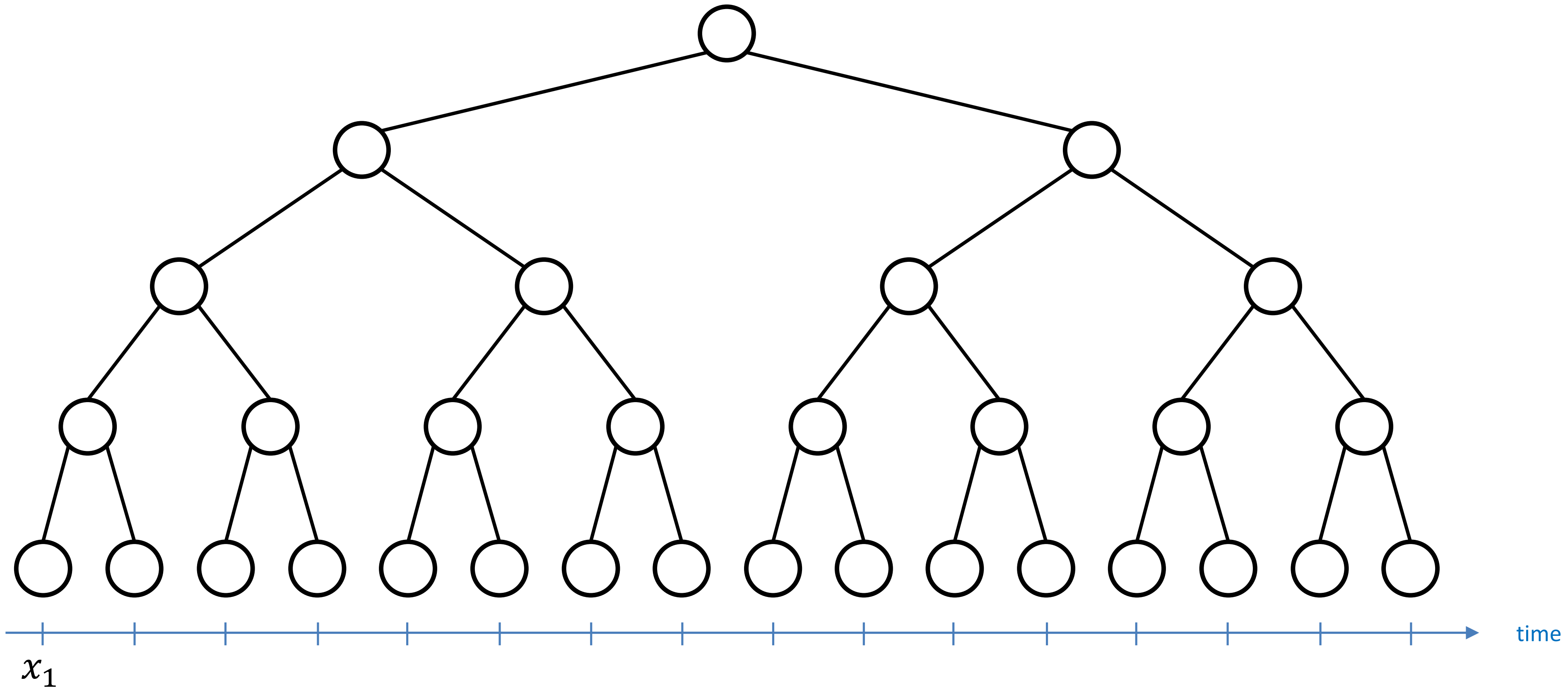
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1},\mathbf{2},\mathbf{3},\ldots,\boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

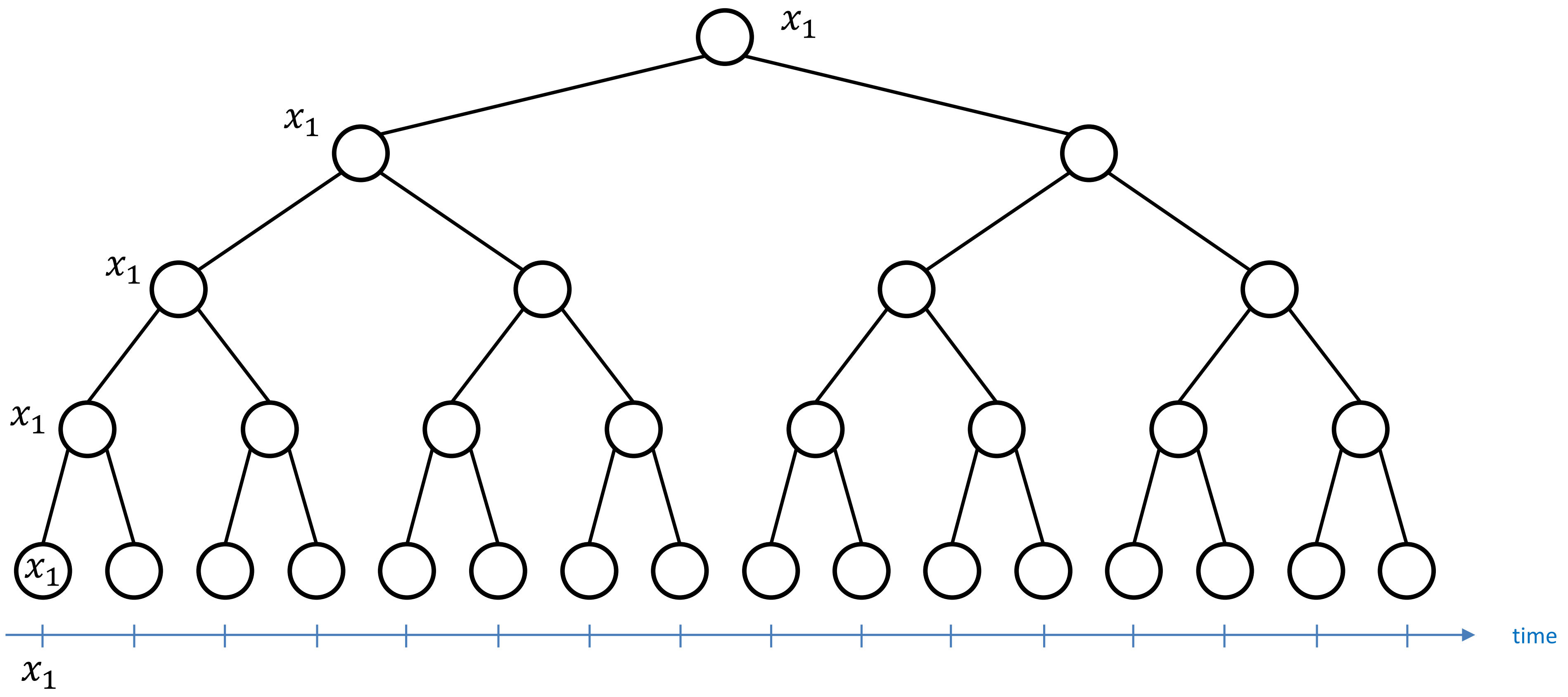
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

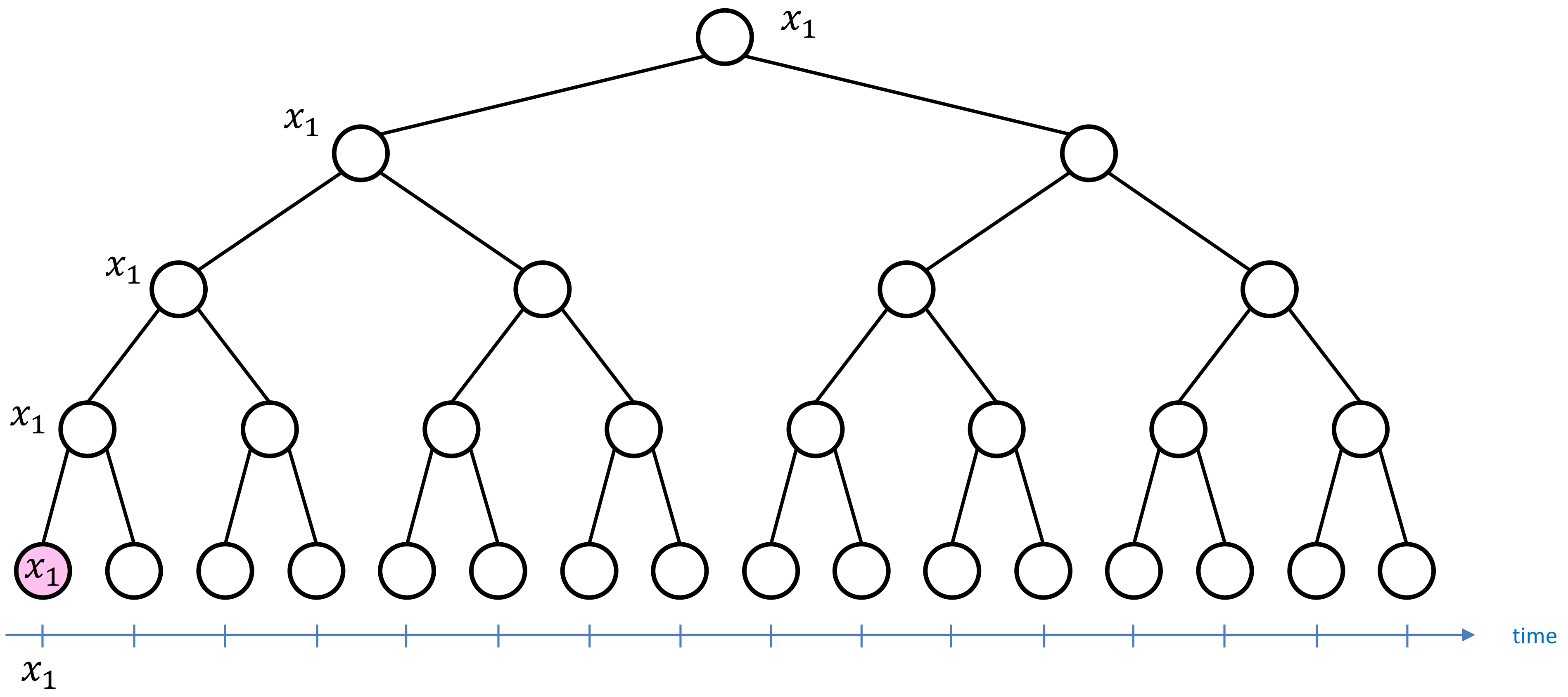
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

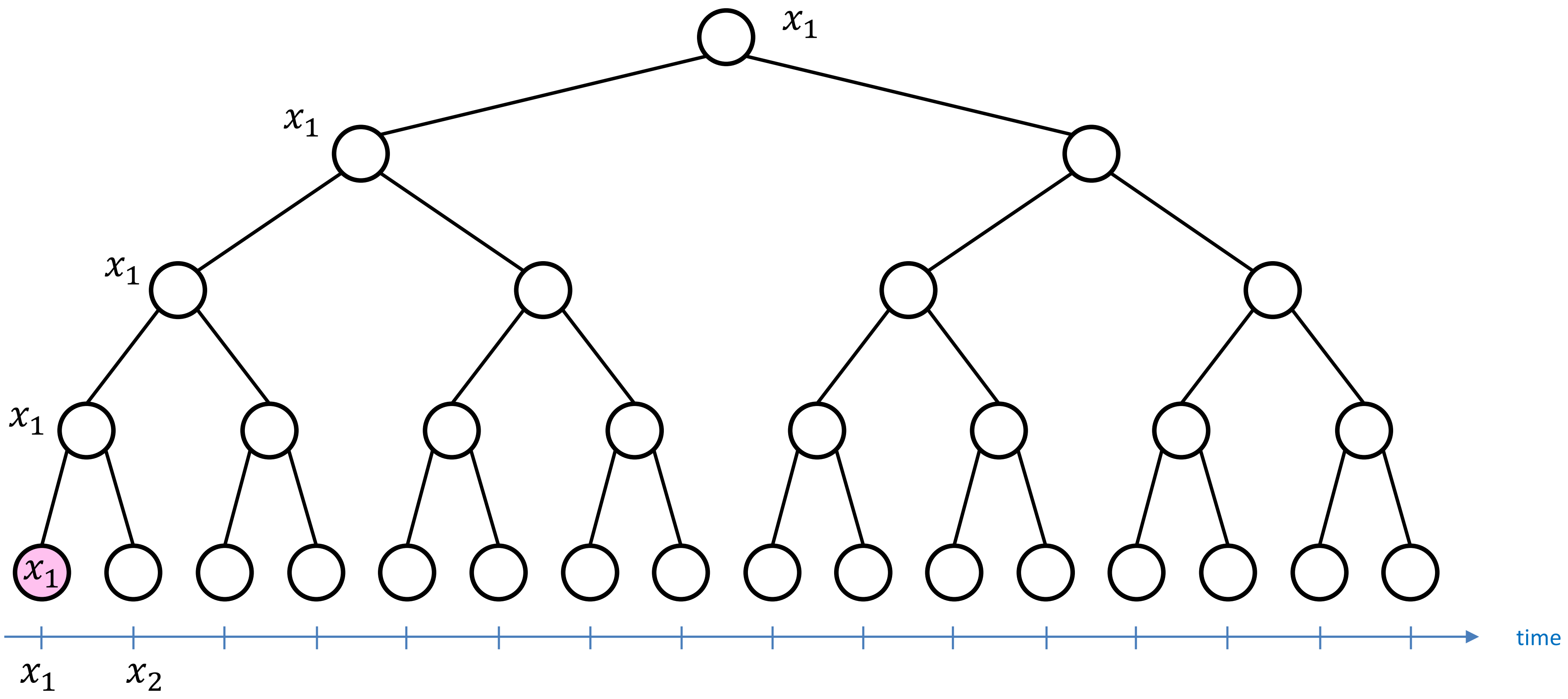
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is “full” we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

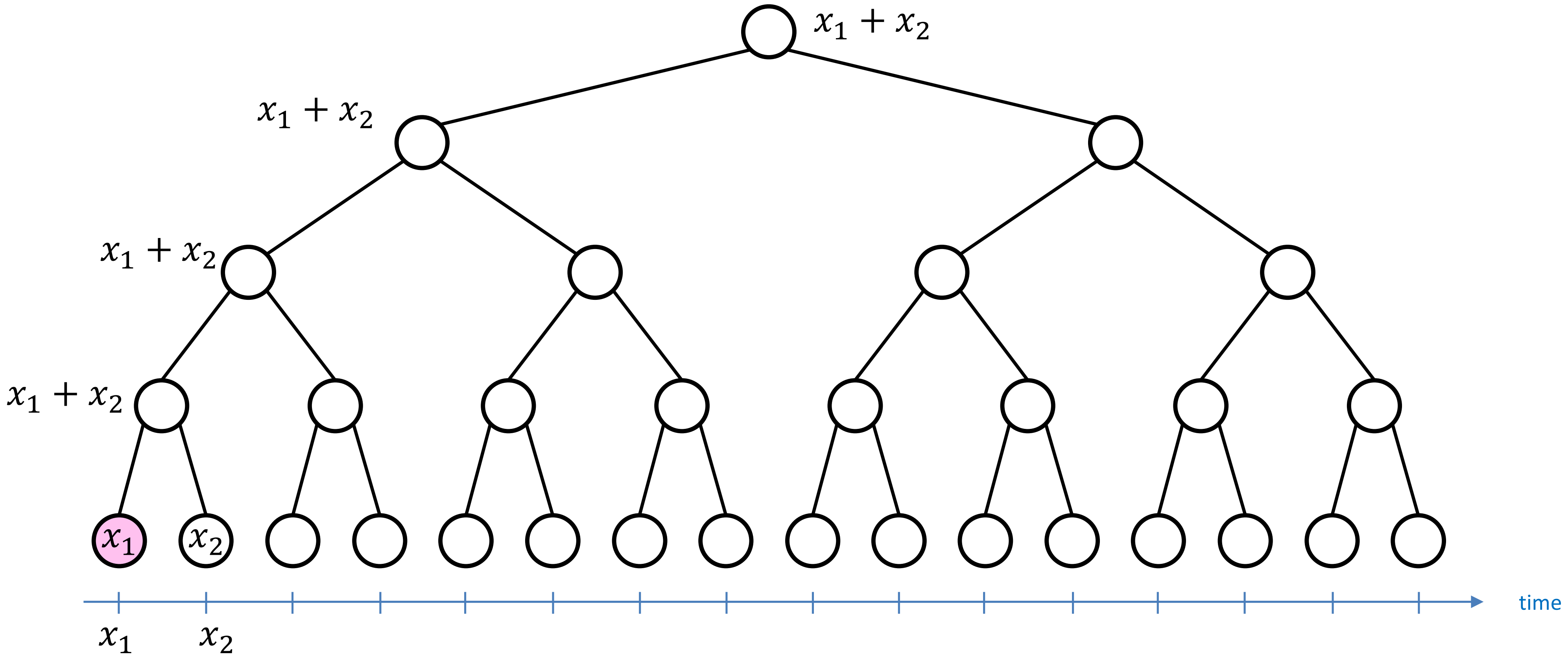
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1},\mathbf{2},\mathbf{3},\dots,\boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is “full” we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

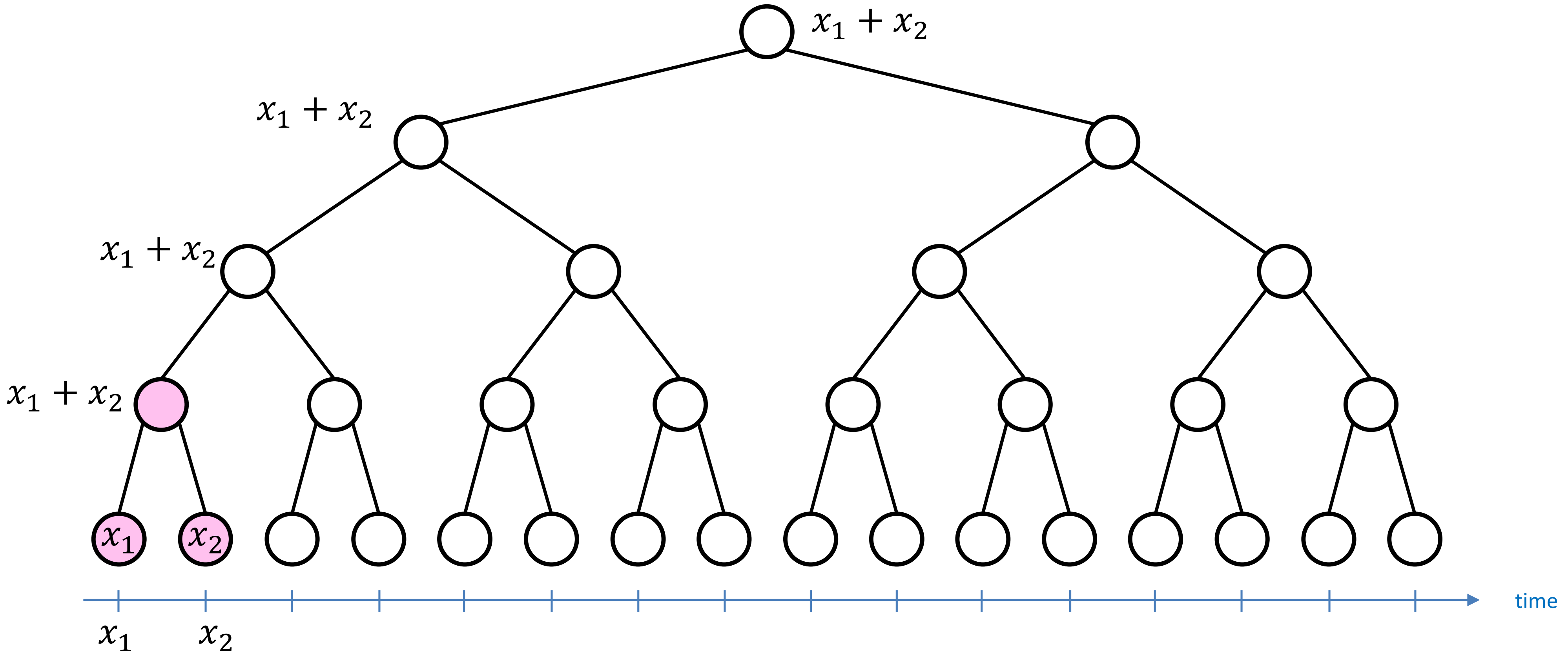
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1},\mathbf{2},\mathbf{3},\dots,\boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

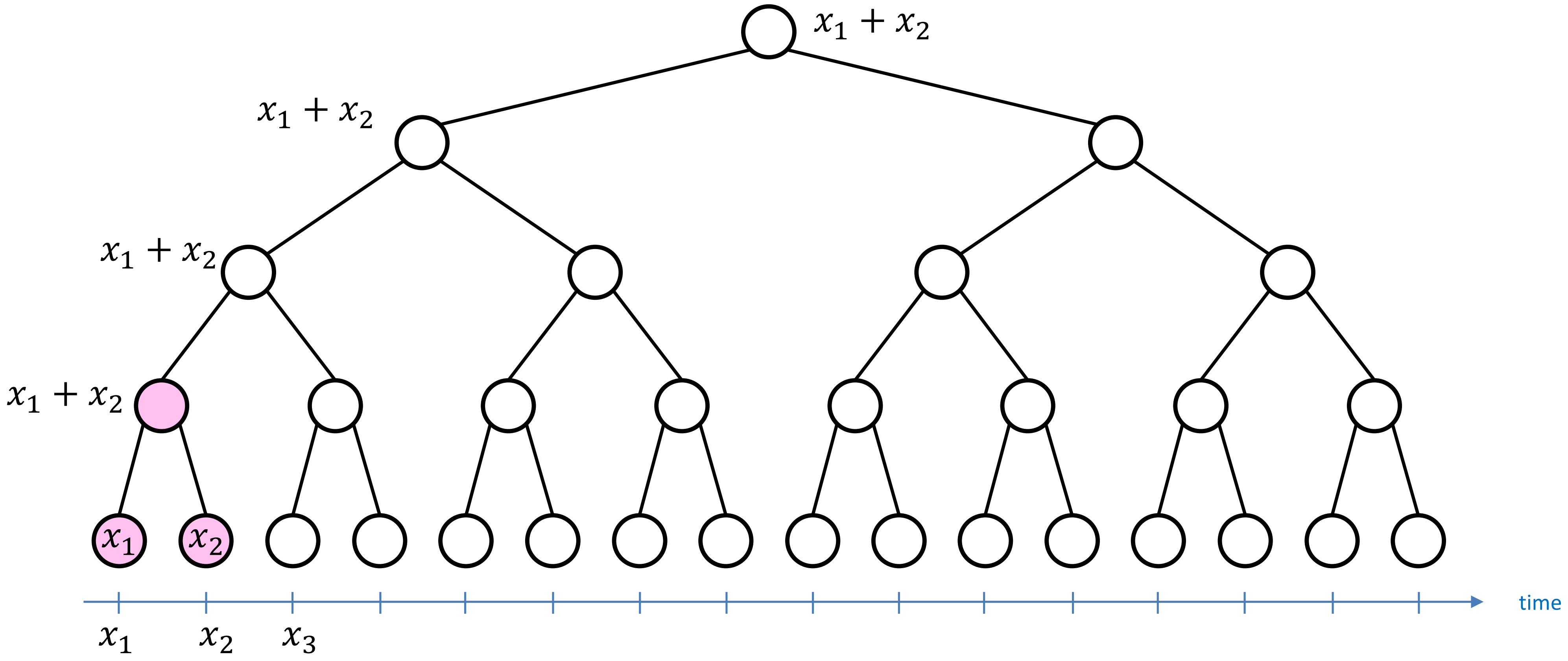
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

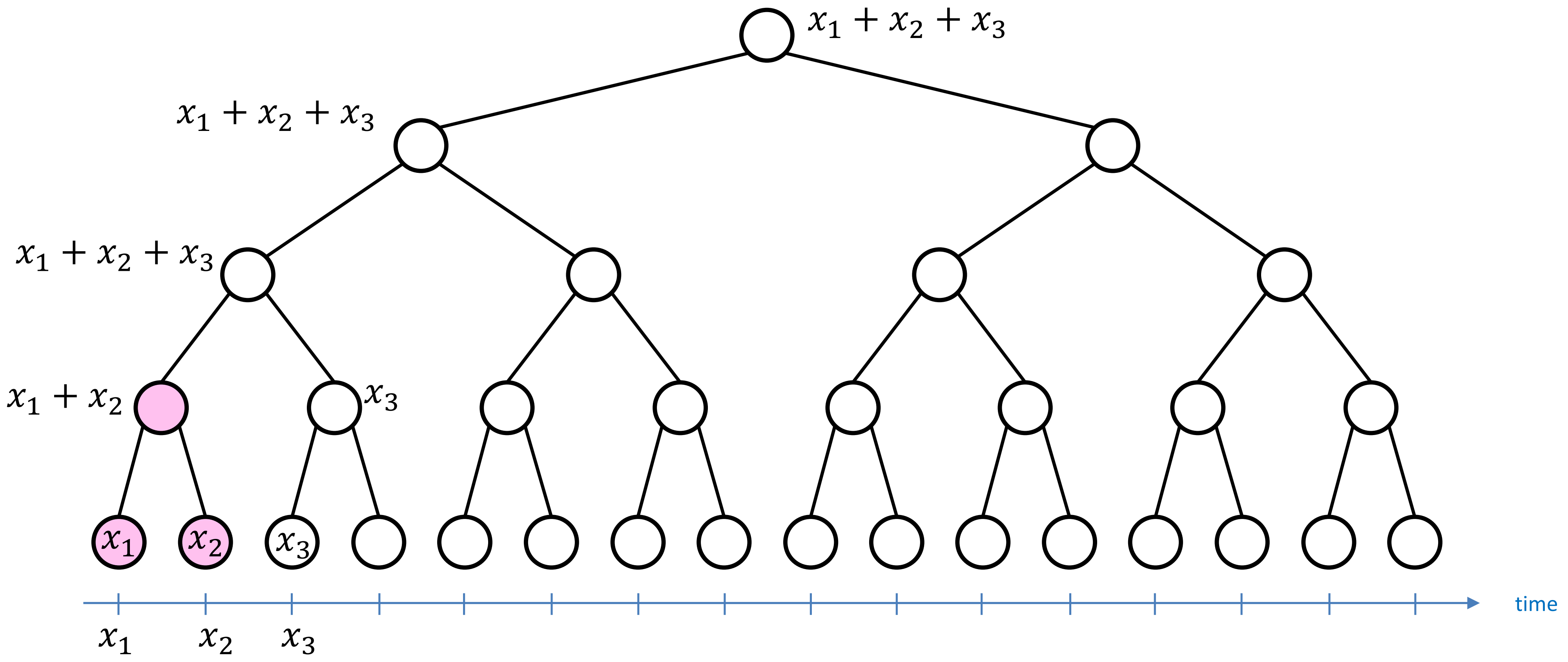
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

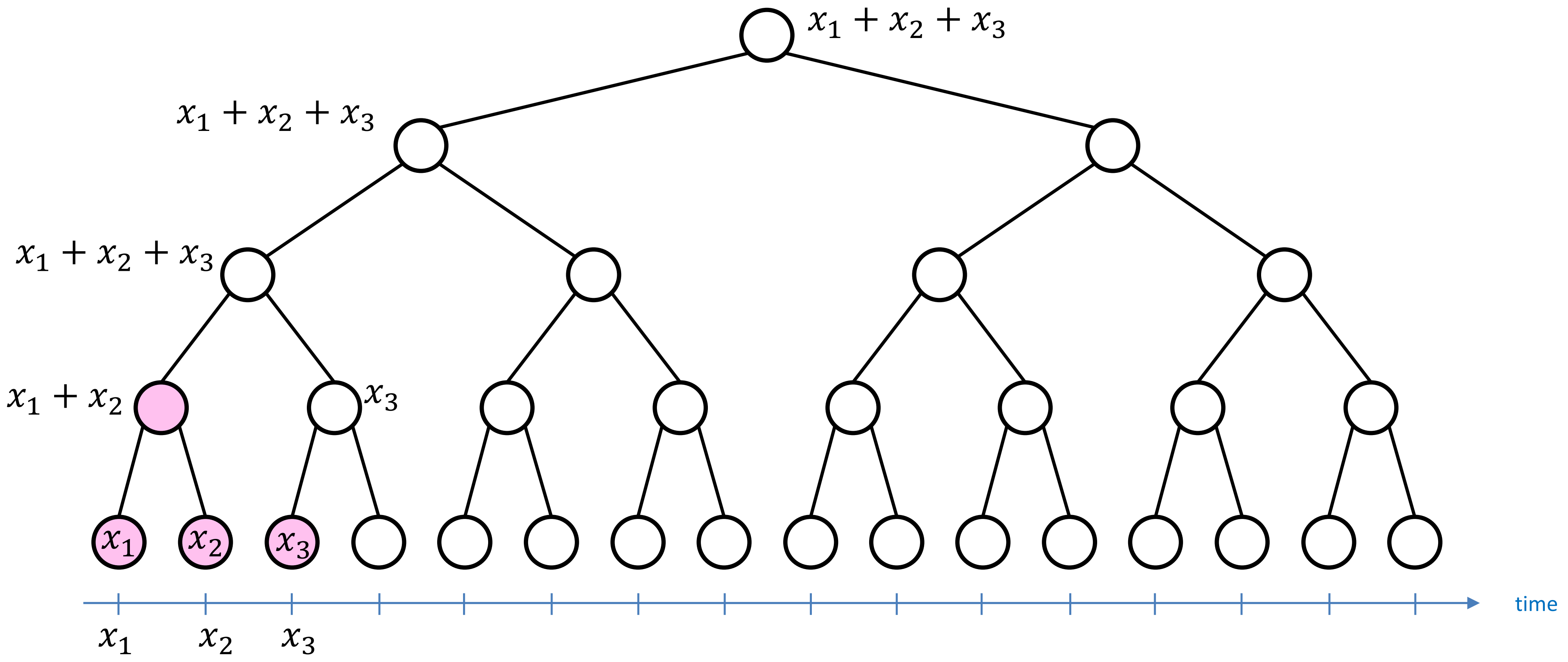
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

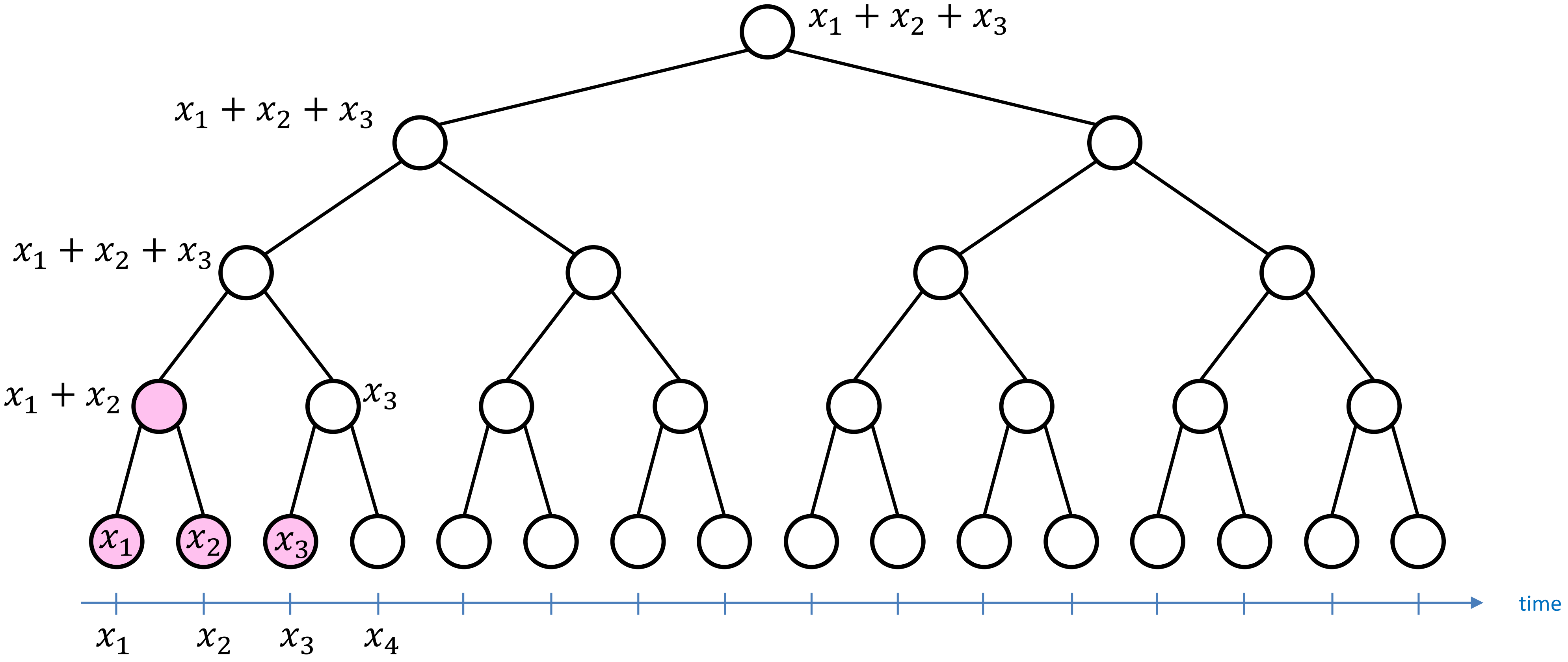
1) Define a binary tree whose leaves correspond to time steps $\{1, 2, 3, \dots, n\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time t we get x_t and add it to all the nodes along the path from leaf t till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

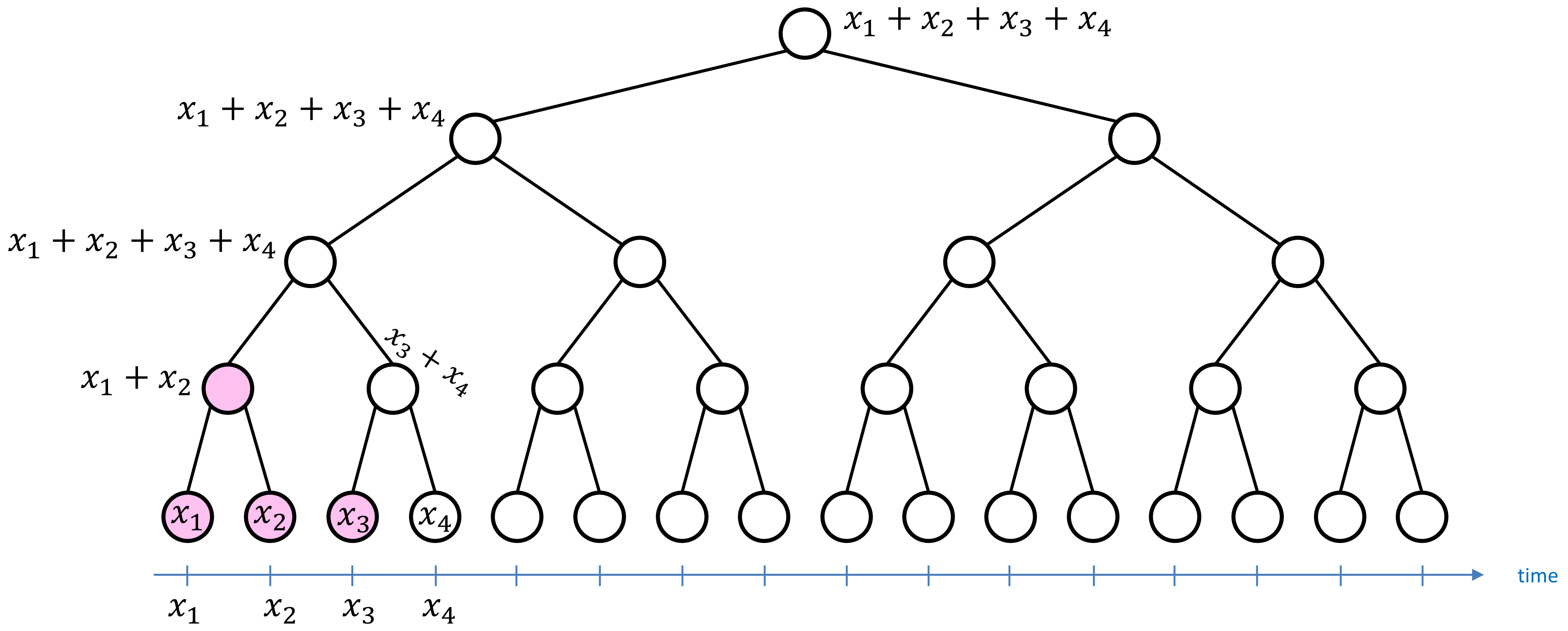
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1, 2, 3, \dots, n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

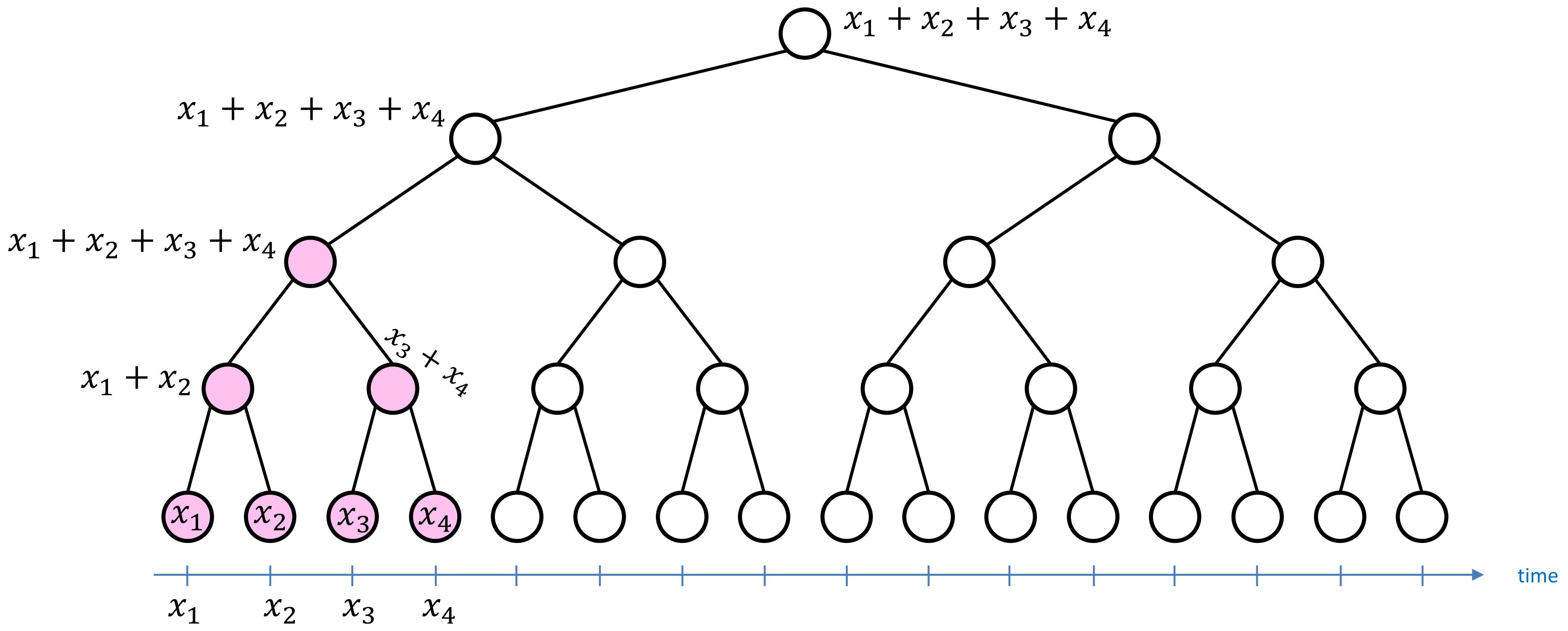
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1},\mathbf{2},\mathbf{3},\dots,\boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

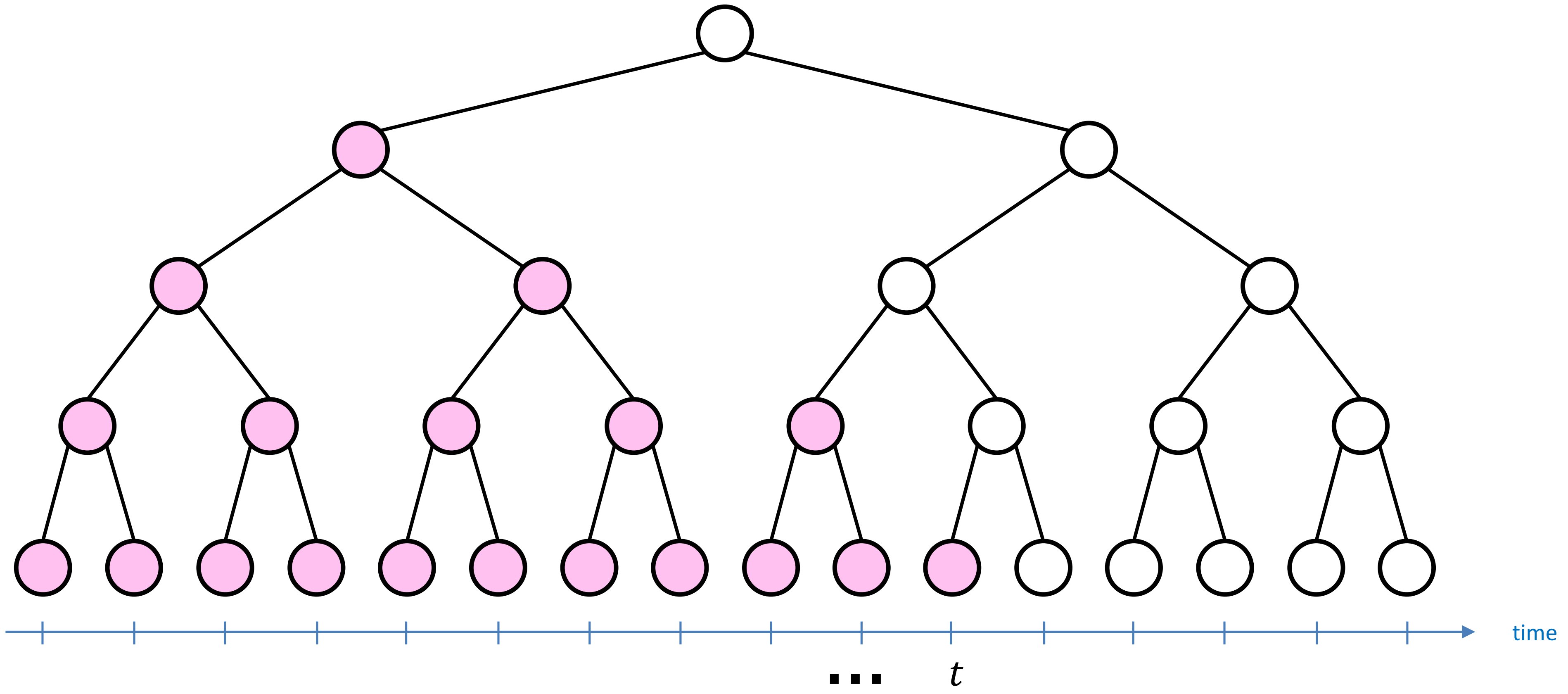
1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1},\mathbf{2},\mathbf{3},\ldots,\boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\mathbf{log}\,\boldsymbol{n}}{\boldsymbol{\varepsilon}}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root

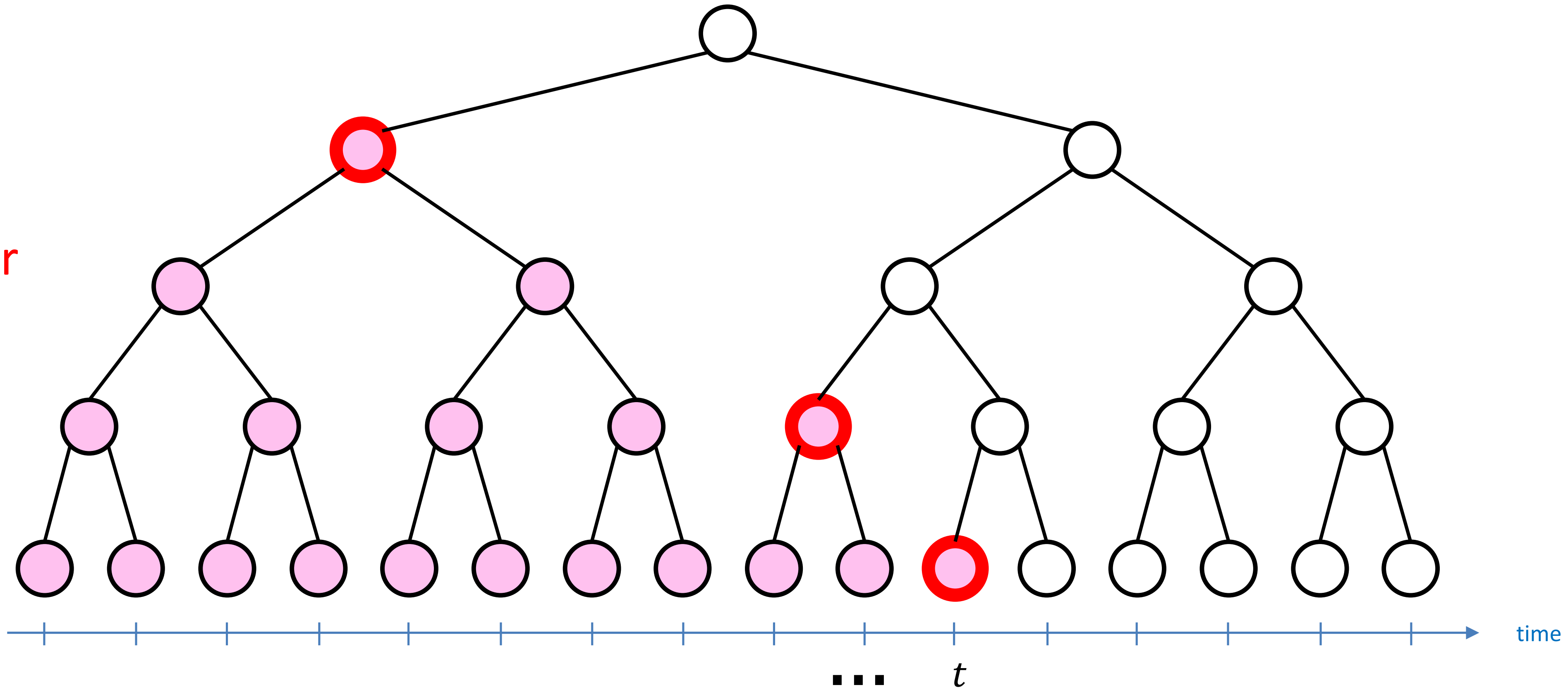


Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root

In every time step we can estimate the current counter by summing at most $\mathbf{log}\,\boldsymbol{n}$ noisy vertices



Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log \boldsymbol{n}}{\boldsymbol{\varepsilon}}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is “full” we release the content of its root

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log \boldsymbol{n}}{\boldsymbol{\varepsilon}}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is “full” we release the content of its root

Privacy analysis:

- Once a subtree is “full” then its root is never updated again
- Thus, we release the content of every node exactly once after adding Laplace noise
- Changing one $\boldsymbol{x_t}$ affects only $\log m$ of these noise, so noise $\frac{\log \boldsymbol{n}}{\boldsymbol{\varepsilon}}$ suffices “by composition”

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

1) Define a binary tree whose leaves correspond to time steps $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{n}\}$
2) We initialize every node with independent random noise from $\mathbf{Lap}\left(\frac{\log n}{\varepsilon}\right)$
3) In time $\boldsymbol{t}$ we get $\boldsymbol{x_t}$ and add it to all the nodes along the path from leaf $\boldsymbol{t}$ till the root
4) When a subtree is "full" we release the content of its root

Privacy analysis:

- Once a subtree is "full" then its root is never updated again
- Thus, we release the content of every node exactly once after adding Laplace noise
- Changing one $\boldsymbol{x_t}$ affects only $\log m$ of these noise, so noise $\frac{\log n}{\varepsilon}$ suffices "by composition"

Utility analysis:

- At any time $\boldsymbol{t}$ we can compute an estimated counter by summing at most $\mathbf{log}\, \boldsymbol{n}$ nodes
- So we are only summing $\mathbf{log}\, \boldsymbol{n}$ noises, each of magnitude $\approx \mathbf{log}\, \boldsymbol{n}$
- Overall error is $\frac{\text{polylog}\, n}{\varepsilon}$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

<u>Thm:</u> Every $(1, 0)$-DP algorithm for this problem must have error $\Omega(\log n)$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Thm: Every $(1, 0)$-DP algorithm for this problem must have error $\Omega(\log n)$

- Suppose there is a private algorithm $\mathcal{A}$ such that w.p. $2/3$ all of its estimates are accurate to within error $\frac{\log n}{16}$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Thm: Every $(1, 0)$-DP algorithm for this problem must have error $\Omega(\log n)$

- Suppose there is a private algorithm $\mathcal{A}$ such that w.p. $2/3$ all of its estimates are accurate to within error $\frac{\log n}{16}$
- Construct a collection $H = \left\{\vec{x}^{(1)}, \vec{x}^{\left(\frac{\log n}{4}\right)}, \dots\right\}$ of input sequences where $\vec{x}^{(i)} = \left(0, 0, \dots, 0, \underbrace{1, 1, 1, \dots, 1}_{\frac{\log n}{4} \text{ ones from time } i}, 0, \dots, 0\right)$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

<u>Thm:</u> Every $(\mathbf{1},\mathbf{0})$-DP algorithm for this problem must have error $\boldsymbol{\Omega}(\log \boldsymbol{n})$

- Suppose there is a private algorithm $\mathcal{A}$ such that w.p. $\mathbf{2/3}$ all of its estimates are accurate to within error $\frac{\log n}{16}$
- Construct a collection $H = \left\{\vec{x}^{(1)}, \vec{x}^{\left(\frac{\log n}{4}\right)}, \dots\right\}$ of input sequences where $\vec{x}^{(i)} = \Big(0,0,\dots,0, \underbrace{1,1,1,\dots,1}_{\frac{\log n}{4}\text{ ones from time } i}, 0,\dots,0\Big)$
- The algorithm has error at most $\frac{\log n}{16}$ so if we run it on input $\vec{x}^{(i)}$ then:
  - Before time i the estimation must be at most $\frac{\log n}{16}$
  - After time $i + \frac{\log n}{4}$ the estimation must be at least $\frac{3\log n}{16}$
- Hence, a sequence of answers cannot by good for more than one $\vec{x}^{(i)}$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Thm: Every $(\mathbf{1},\mathbf{0})$-DP algorithm for this problem must have error $\boldsymbol{\Omega}(\log \boldsymbol{n})$

- Suppose there is a private algorithm $\boldsymbol{\mathcal{A}}$ such that w.p. $\mathbf{2/3}$ all of its estimates are accurate to within error $\frac{\log n}{16}$
- Construct a collection $\boldsymbol{H} = \left\{\vec{\boldsymbol{x}}^{(\mathbf{1})}, \vec{\boldsymbol{x}}^{\left(\frac{\log n}{4}\right)}, \ldots\right\}$ of input sequences where $\vec{x}^{(i)} = \left(\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}, \underbrace{\mathbf{1}, \mathbf{1}, \mathbf{1}, \ldots, \mathbf{1}}_{\frac{\log n}{4} \text{ ones from time } i}, \mathbf{0}, \ldots, \mathbf{0}\right)$
- The algorithm has error at most $\frac{\log n}{16}$ so if we run it on input $\vec{\boldsymbol{x}}^{(\boldsymbol{i})}$ then:
  - Before time $\boldsymbol{i}$ the estimation must be at most $\frac{\log n}{16}$
  - After time $\boldsymbol{i} + \frac{\log n}{4}$ the estimation must be at least $\frac{3\log n}{16}$
- Hence, a sequence of answers cannot by good for more than one $\vec{\boldsymbol{x}}^{(\boldsymbol{i})}$ and for every $\boldsymbol{i} \neq \ell$ we have

$$\frac{2}{3} \leq \Pr\left[\begin{matrix}\mathcal{A}\left(\vec{x}^{(i)}\right) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{matrix}\right] \leq e^{2\frac{\log n}{4}} \cdot \Pr\left[\begin{matrix}\mathcal{A}\left(\vec{x}^{(\ell)}\right) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{matrix}\right] = \sqrt{n} \cdot \Pr\left[\begin{matrix}\mathcal{A}\left(\vec{x}^{(\ell)}\right) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{matrix}\right]$$

Private counter under continual observation

[Dwork, Naor, Pitassi, Rothblum]

Thm: Every $(1,0)$-DP algorithm for this problem must have error $\Omega(\log n)$

- Suppose there is a private algorithm $\mathcal{A}$ such that w.p. $2/3$ all of its estimates are accurate to within error $\frac{\log n}{16}$
- Construct a collection $H = \left\{\vec{x}^{(1)}, \vec{x}^{\left(\frac{\log n}{4}\right)}, \dots\right\}$ of input sequences where $\vec{x}^{(i)} = \left(0,0,\dots,0, \underbrace{1,1,1,\dots,1}_{\frac{\log n}{4}\text{ ones from time } i}, 0,\dots,0\right)$
- The algorithm has error at most $\frac{\log n}{16}$ so if we run it on input $\vec{x}^{(i)}$ then:
  - Before time i the estimation must be at most $\frac{\log n}{16}$
  - After time $i + \frac{\log n}{4}$ the estimation must be at least $\frac{3\log n}{16}$
- Hence, a sequence of answers cannot by good for more than one $\vec{x}^{(i)}$ and for every $i \neq \ell$ we have

$$\frac{2}{3} \le \Pr\begin{bmatrix}\mathcal{A}\left(\vec{x}^{(i)}\right)\text{ returns}\\ \text{good answers}\\ \text{for } \vec{x}^{(i)}\end{bmatrix} \le e^{2\frac{\log n}{4}} \cdot \Pr\begin{bmatrix}\mathcal{A}\left(\vec{x}^{(\ell)}\right)\text{ returns}\\ \text{good answers}\\ \text{for } \vec{x}^{(i)}\end{bmatrix} = \sqrt{n} \cdot \Pr\begin{bmatrix}\mathcal{A}\left(\vec{x}^{(\ell)}\right)\text{ returns}\\ \text{good answers}\\ \text{for } \vec{x}^{(i)}\end{bmatrix}$$

- So,

$$\Pr\begin{bmatrix}\mathcal{A}\left(\vec{x}^{(\ell)}\right)\text{ returns}\\ \text{good answers}\\ \text{for } \vec{x}^{(i)}\end{bmatrix} \ge \frac{2}{3\sqrt{n}}$$

$$\Pr[\mathcal{A}(\vec{x}^{(\ell)}) \text{ fails}] \geq \Pr\left[\bigcup_{i\neq\ell}\left\{\begin{matrix}\mathcal{A}(\vec{x}^{(\ell)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{matrix}\right\}\right] \underbrace{=}_{\substack{\text{disjoint}\\ \text{events}}} \sum_{i\neq\ell} \Pr\begin{bmatrix}\mathcal{A}(\vec{x}^{(\ell)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{bmatrix} \gtrsim \frac{n}{\log n}\cdot\frac{1}{\sqrt{n}} > 1$$

- Suppose there is a private algorithm $\boldsymbol{\mathcal{A}}$ such that w.p. $\mathbf{2/3}$ all of its estimates are accurate to within error $\frac{\log n}{16}$
- Construct a collection $\boldsymbol{H} = \left\{\vec{\boldsymbol{x}}^{(1)}, \vec{\boldsymbol{x}}^{\left(\frac{\log n}{4}\right)}, \dots\right\}$ of input sequences where $\vec{\boldsymbol{x}}^{(i)} = \left(\mathbf{0}, \mathbf{0}, \dots, \mathbf{0}, \underbrace{\mathbf{1}, \mathbf{1}, \mathbf{1}, \dots, \mathbf{1}}_{\frac{\log n}{4} \text{ ones from time } i}, \mathbf{0}, \dots, \mathbf{0}\right)$
- The algorithm has error at most $\frac{\log n}{16}$ so if we run it on input $\vec{\boldsymbol{x}}^{(i)}$ then:
  - Before time $\boldsymbol{i}$ the estimation must be at most $\frac{\log n}{16}$
  - After time $\boldsymbol{i} + \frac{\log n}{4}$ the estimation must be at least $\frac{3\log n}{16}$
- Hence, a sequence of answers cannot by good for more than one $\vec{\boldsymbol{x}}^{(i)}$ and for every $\boldsymbol{i} \neq \ell$ we have

$$\frac{2}{3} \leq \Pr\begin{bmatrix}\mathcal{A}(\vec{x}^{(i)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{bmatrix} \leq e^{2\frac{\log n}{4}} \cdot \Pr\begin{bmatrix}\mathcal{A}(\vec{x}^{(\ell)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{bmatrix} = \sqrt{n} \cdot \Pr\begin{bmatrix}\mathcal{A}(\vec{x}^{(\ell)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{bmatrix}$$

- So,

$$\Pr\begin{bmatrix}\mathcal{A}(\vec{x}^{(\ell)}) \text{ returns} \\ \text{good answers} \\ \text{for } \vec{x}^{(i)}\end{bmatrix} \geq \frac{2}{3\sqrt{n}}$$

Negative result for continual observation

[Jain, Raskhodnikova, Sivakumar, Smith 2021]

Recall the XOR-sum problem:

- The input of every user i is a pair $(j_i, b_i) \in \{1,2,\ldots,n\} \times \{0,1\}$
- The goal: estimate $\sum_{j=1}^{n} \bigoplus_{i:j_i=j} b_i$

Negative result for continual observation

[Jain, Raskhodnikova, Sivakumar, Smith 2021]

Recall the XOR-sum problem:

- The input of every user i is a pair $(j_i, b_i) \in \{1,2,\dots,n\} \times \{0,1\}$
- The goal: estimate $\sum_{j=1}^{n} \bigoplus_{i:j_i=j} b_i$

As we mentioned: Can be solved with error $\approx \frac{1}{\varepsilon}$ in the centralized model

Informal theorem: Any continual-DP algorithm for this problem must have error $\Omega(\text{poly}\, n)$

Negative result for continual observation

[Jain, Raskhodnikova, Sivakumar, Smith 2021]

Recall the XOR-sum problem:

- The input of every user i is a pair $(j_i, b_i) \in \{1,2,\dots,n\} \times \{0,1\}$
- The goal: estimate $\sum_{j=1}^{n} \bigoplus_{i:j_i=j} b_i$

As we mentioned: Can be solved with error $\approx \frac{1}{\varepsilon}$ in the centralized model

Informal theorem: Any continual-DP algorithm for this problem must have error $\Omega(\text{poly}\, n)$

Proof:

- An algorithm $\mathcal{A}$ for this problem can be used to answer **many** "Hamming distance queries"; contradiction…

Negative result for continual observation

[Jain, Raskhodnikova, Sivakumar, Smith 2021]

Recall the XOR-sum problem:

- The input of every user i is a pair $(j_i, b_i) \in \{1,2,\ldots,n\} \times \{0,1\}$
- The goal: estimate $\sum_{j=1}^{n} \bigoplus_{i:j_i=j} b_i$

As we mentioned: Can be solved with error $\approx \frac{1}{\varepsilon}$ in the centralized model

Informal theorem: Any continual-DP algorithm for this problem must have error $\Omega(\text{poly}\, n)$

Proof:

- An algorithm $\mathcal{A}$ for this problem can be used to answer **many** "Hamming distance queries"; contradiction…
- Specifically, given input dataset $\boldsymbol{X} = \left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\sqrt{n}}\right)$, feed $(\mathbf{1}, \boldsymbol{x}_1), \ldots, \left(\sqrt{\boldsymbol{n}}, \boldsymbol{x}_{\sqrt{n}}\right)$ to algorithm $\mathcal{A}$
- Then, given a query $\boldsymbol{Y} = \left(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\sqrt{n}}\right)$:
    - Feed $(\mathbf{1}, \boldsymbol{y}_1), \ldots, \left(\sqrt{\boldsymbol{n}}, \boldsymbol{y}_{\sqrt{n}}\right)$ to algorithm $\mathcal{A}$ to obtain an answer $\boldsymbol{z}$
    - Feed $(\mathbf{1}, \boldsymbol{y}_1), \ldots, \left(\sqrt{\boldsymbol{n}}, \boldsymbol{y}_{\sqrt{n}}\right)$ to algorithm $\mathcal{A}$ again
- After $\approx \sqrt{\boldsymbol{n}}$ queries (total input length $\boldsymbol{n}$) we can reconstruct $\boldsymbol{X}$ contradicting the privacy of $\mathcal{A}$

Streaming/online settings

Today's Outline

✓ 1. Private streaming algorithms

✓ 2. Privacy under continual observation