Modular Verification of Differential Privacy in Probabilistic Higher-Order Separation Logic

Philipp G. Haselwarter[1], Alejandro Aguirre[1], Simon Oddershede Gregersen[2], Kwing Hei Li[1], Joseph Tassarotti[3], and Lars Birkedal[1]

June 10, 2026

[1] *Aarhus University,* [2] *CISPA,* [3] *New York University*

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential privacy solution:

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential privacy solution:

Release statistics without revealing *whether Frank is in* D.

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential privacy solution:

Release statistics without revealing *whether Frank is in* D.

Define a hypothetical $D' := \{\text{rows in } D \text{ with Frank's row removed}\}$.

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential privacy solution:

Release statistics without revealing *whether Frank is in* D.

Define a hypothetical $D' := \{\text{rows in } D \text{ with Frank's row removed}\}$.

$D \sim D'$ the two datasets are "*adjacent*".

Differential Privacy: Goal

Sensitive dataset D:

name	age	ZIP	BMI	specialty
Alice	31	8000	26	dental
Bob	45	1050	24	neurology
Frank	**52**	**2100**	**25**	**cardiology**
Carol	27	8200	28	general practice
⋮	⋮	⋮	⋮	⋮

We want to release statistics $f(D)$ "without revealing too much about Frank".

Differential privacy solution:

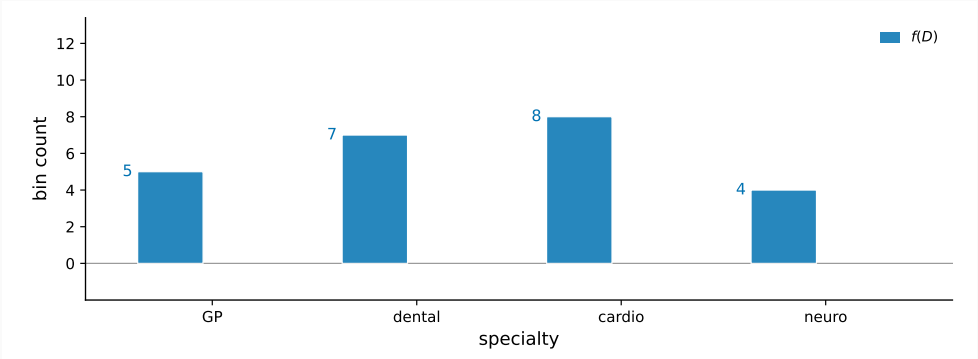
Release statistics without revealing *whether Frank is in* D.

Define a hypothetical $D' := \{\text{rows in } D \text{ with Frank's row removed}\}$.

$D \sim D'$ the two datasets are "*adjacent*".

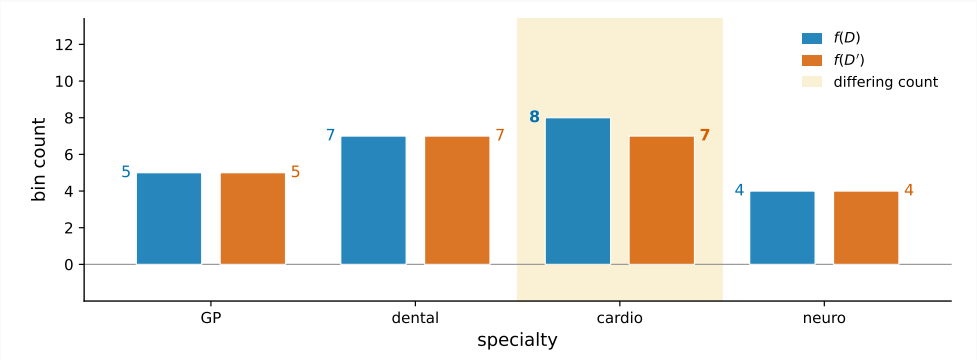
Is $f(D)$ "similar to" $f(D')$?

Differential Privacy: Method



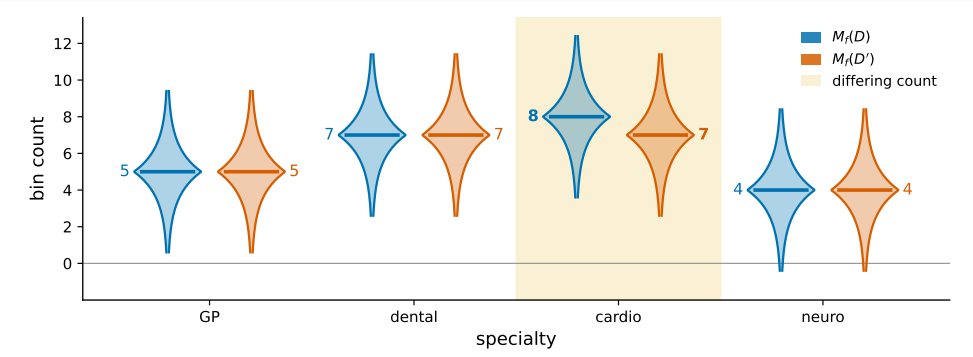
- f computes a histogram: count #members in each specialty.

Differential Privacy: Method



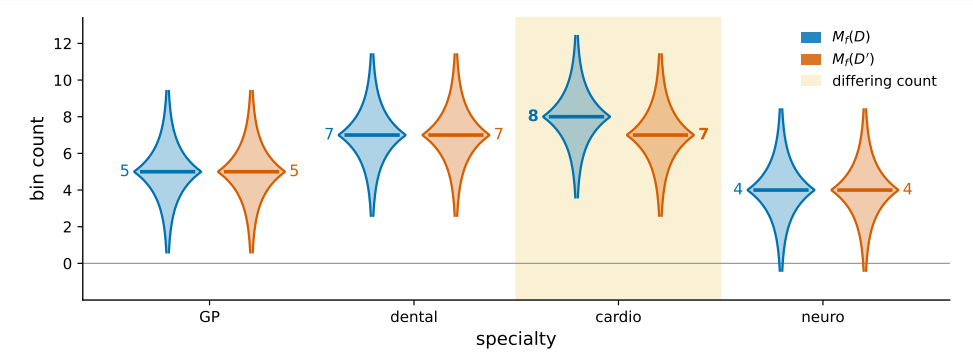
- f computes a histogram: count #members in each specialty.
- $f(D)$ and $f(D')$ differ only in Frank's specialty: easy to read Frank's data off the histogram.

Differential Privacy: Method



- f computes a histogram: count #members in each specialty.
- $f(D)$ and $f(D')$ differ only in Frank's specialty: easy to read Frank's data off the histogram.
- DP "*mechanism*" $M_f(-)$ **adds random noise** to each count, releasing *distributions*.

Differential Privacy: Method



- f computes a histogram: count #members in each specialty.
- $f(D)$ and $f(D')$ differ only in Frank's specialty: easy to read Frank's data off the histogram.
- DP "*mechanism*" $M_f(-)$ **adds random noise** to each count, releasing *distributions*.
  $\Rightarrow$ Samples drawn from $M_f(D)$ and $M_f(D')$ **probably look similar** to an observer.

From a noisy bin to industrial DP code

```
def histogram(D):
    ...
    count = sum(x.cat==c for x in D)
    result[c] = count
    ...
    return result
```

From a noisy bin to industrial DP code

```
def histogram(D):
    ...
    count = sum(x.cat==c for x in D)
    result[c] = count
    ...
    return result
```

```
def dp_histogram(D, eps):
    ...
    count = sum(x.cat==c for x in D)
    noisy = count + Lap(1/eps)
    result[c] = noisy
    ...
    return result
```

From a noisy bin to industrial DP code

```
def histogram(D):
    ...
    count = sum(x.cat==c for x in D)
    result[c] = count
    ...
    return result
```

```
def dp_histogram(D, eps):
    ...
    count = sum(x.cat==c for x in D)
    noisy = count + Lap(1/eps)
    result[c] = noisy
    ...
    return result
```

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget
  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

From a noisy bin to industrial DP code

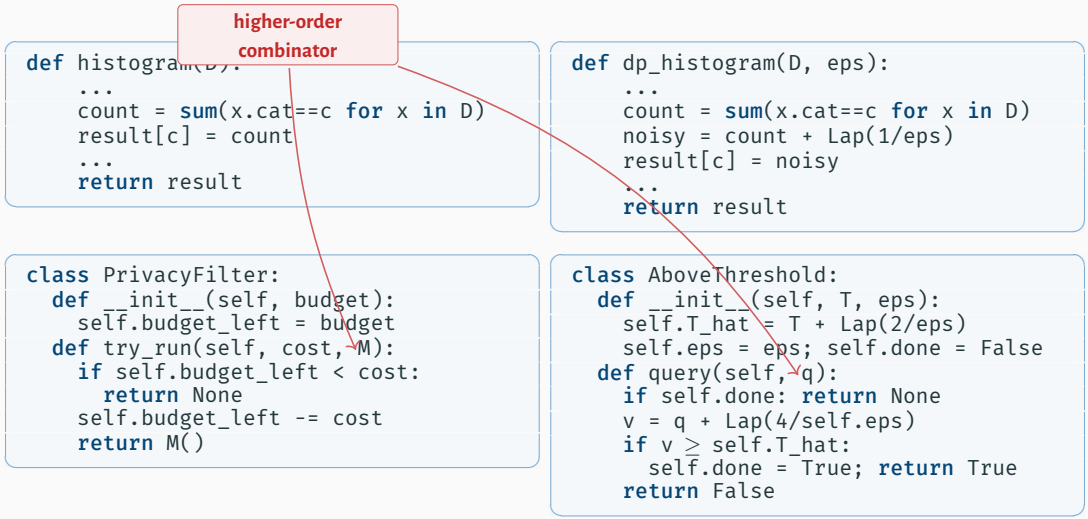
```
def histogram(D):
    ...
    count = sum(x.cat==c for x in D)
    result[c] = count
    ...
    return result
```

```
def dp_histogram(D, eps):
    ...
    count = sum(x.cat==c for x in D)
    noisy = count + Lap(1/eps)
    result[c] = noisy
    ...
    return result
```

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget
  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

```
class AboveThreshold:
  def __init__(self, T, eps):
    self.T_hat = T + Lap(2/eps)
    self.eps = eps; self.done = False
  def query(self, q):
    if self.done: return None
    v = q + Lap(4/self.eps)
    if v ≥ self.T_hat:
      self.done = True; return True
    return False
```

From a noisy bin to industrial DP code



From a noisy bin to industrial DP code

higher-order combinator

local mutable state

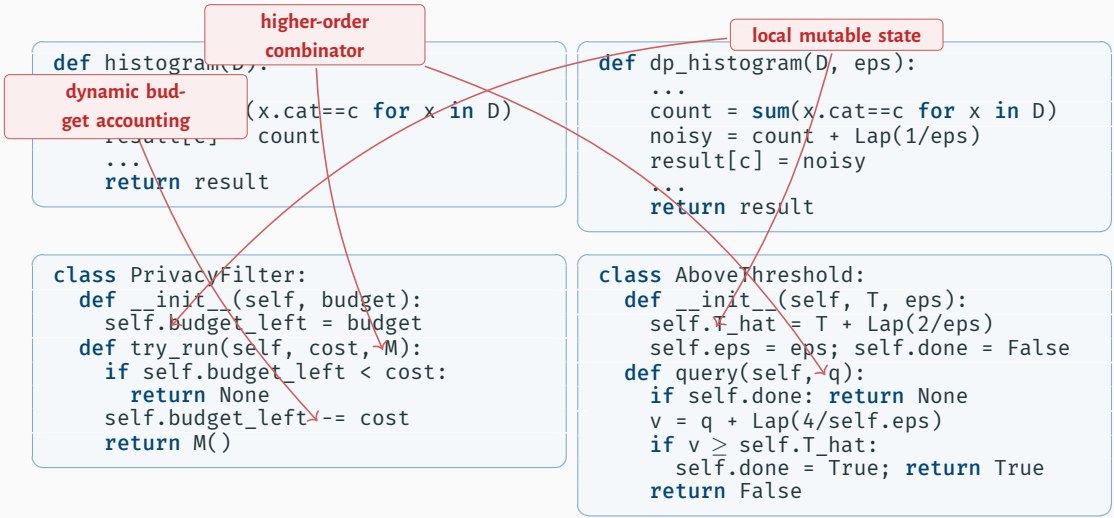
```
def histogram(D):
    ...
    count = sum(x.cat==c for x in D)
    result[c] = count
    ...
    return result
```

```
def dp_histogram(D, eps):
    ...
    count = sum(x.cat==c for x in D)
    noisy = count + Lap(1/eps)
    result[c] = noisy
    ...
    return result
```

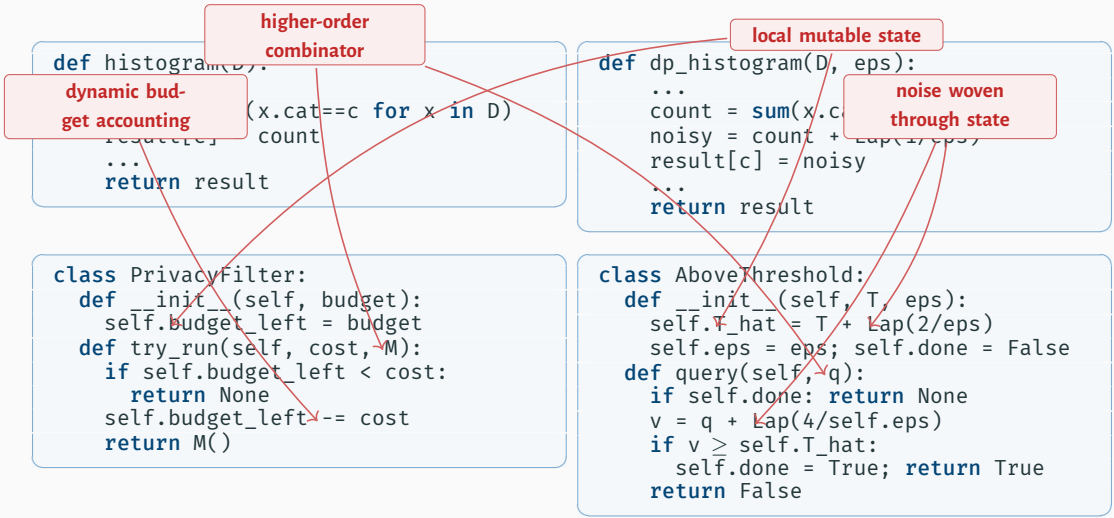
```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget
  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

```
class AboveThreshold:
  def __init__(self, T, eps):
    self.T_hat = T + Lap(2/eps)
    self.eps = eps; self.done = False
  def query(self, q):
    if self.done: return None
    v = q + Lap(4/self.eps)
    if v ≥ self.T_hat:
      self.done = True; return True
    return False
```

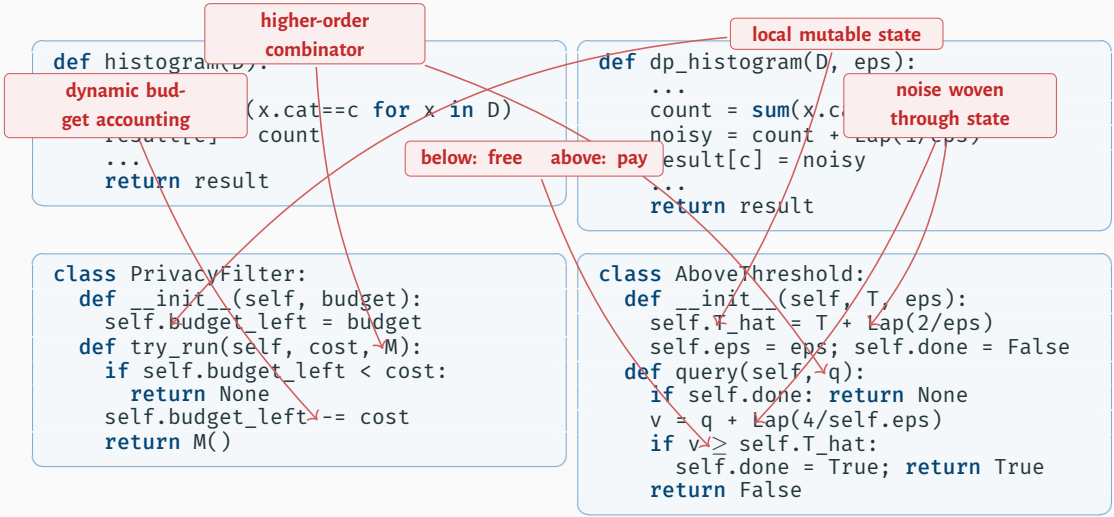
From a noisy bin to industrial DP code



From a noisy bin to industrial DP code



From a noisy bin to industrial DP code



Differential privacy: the definition

(ε, δ)-*differential privacy* Dwork *et al.* 2006

A randomised mechanism $M : \mathsf{DB} \to \mathcal{D}(X)$ is (ε, δ)-**DP** if for every pair of adjacent $D \sim D'$ and every event $S \subseteq X$,

$$\Pr[M(D) \in S] \;\leq\; e^{\varepsilon} \cdot \Pr[M(D') \in S] \;+\; \delta.$$

Differential privacy: the definition

(ε, δ)-*differential privacy* Dwork *et al.* 2006

A randomised mechanism $M\colon \mathsf{DB} \to \mathcal{D}(X)$ is (ε, δ)-**DP** if for every pair of adjacent $D \sim D'$ and every event $S \subseteq X$,

$$\Pr[M(D) \in S] \;\leq\; e^{\varepsilon} \cdot \Pr[M(D') \in S] \;+\; \delta.$$

- e^{ε} bounds the *multiplicative* ratio of probabilities.

Differential privacy: the definition

(ε, δ)-*differential privacy* Dwork *et al.* 2006

A randomised mechanism $M : \mathsf{DB} \to \mathcal{D}(X)$ is (ε, δ)-**DP** if for every pair of adjacent $D \sim D'$ and every event $S \subseteq X$,

$$\Pr[M(D) \in S] \;\leq\; e^{\varepsilon} \cdot \Pr[M(D') \in S] \;+\; \delta.$$

- e^{ε} bounds the *multiplicative* ratio of probabilities.
- δ is a small *additive* slack ($\delta = 0$ for the rest of the talk).

Differential privacy: the definition

(ε, δ)-*differential privacy* Dwork *et al.* 2006

A randomised mechanism $M : \mathsf{DB} \to \mathcal{D}(X)$ is (ε, δ)-**DP** if for every pair of adjacent $D \sim D'$ and every event $S \subseteq X$,

$$\Pr[M(D) \in S] \;\leq\; e^{\varepsilon} \cdot \Pr[M(D') \in S] \;+\; \delta.$$

- e^{ε} bounds the *multiplicative* ratio of probabilities.
- δ is a small *additive* slack ($\delta = 0$ for the rest of the talk).
- Small ε means the two distributions are hard to tell apart.

Differential privacy: the definition

(ε, δ)-*differential privacy* Dwork *et al.* 2006

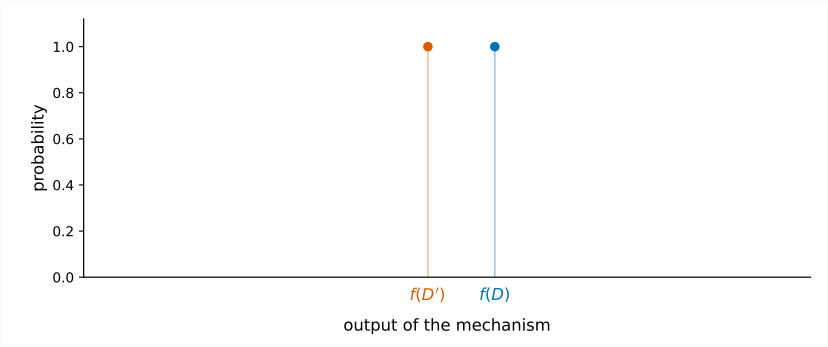
A randomised mechanism $M\colon \mathrm{DB} \to \mathcal{D}(X)$ is (ε, δ)-**DP** if for every pair of adjacent $D \sim D'$ and every event $S \subseteq X$,

$$\Pr[M(D) \in S] \;\leq\; e^{\varepsilon} \cdot \Pr[M(D') \in S] \;+\; \delta.$$

- e^{ε} bounds the *multiplicative* ratio of probabilities.
- δ is a small *additive* slack ($\delta = 0$ for the rest of the talk).
- Small ε means the two distributions are hard to tell apart.

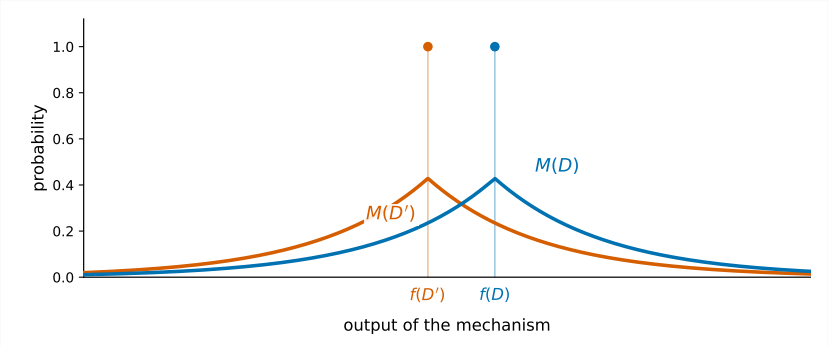
Next: focus on one concrete bin, the category Frank was in.

The Laplace mechanism, visualised



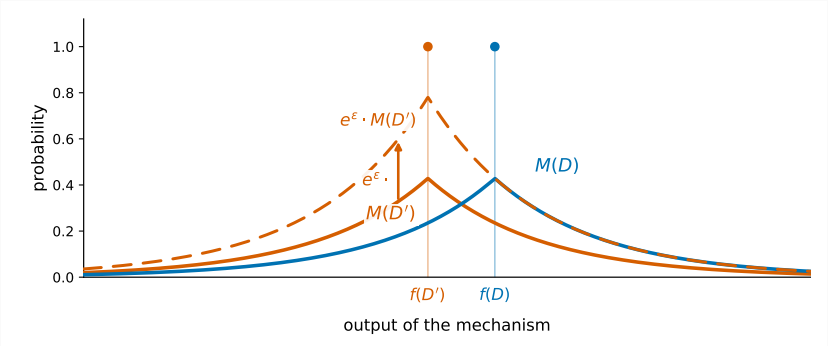
- Bin count x; without noise, M returns the exact answer.

The Laplace mechanism, visualised



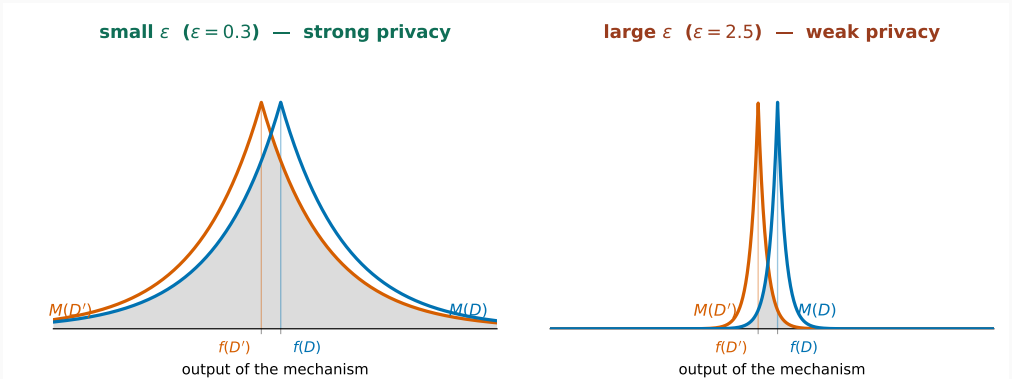
- Bin count x; without noise, M returns the exact answer.
- Add Laplacian noise scaled by ε. Adjacent input shifts the mean of M.

The Laplace mechanism, visualised



- Bin count x; without noise, M returns the exact answer.
- Add Laplacian noise scaled by ε. Adjacent input shifts the mean of M.
- At every outcome, the envelope $e^\varepsilon\ M(D')$ sits above $M(D)$; pure ε-DP, $\delta = 0$.

Calibrating the noise to the analysis



Δ = *sensitivity* of the released function (max change of $f(D)$ under one row change).

ε sets the noise scale $b = \Delta/\varepsilon$; for 1-sensitive functions (histogram counts, etc.) $b = 1/\varepsilon$.

Smaller $\varepsilon \rightarrow$ more noise $\rightarrow$ wider, overlapping releases $\rightarrow$ stronger privacy.

The privacy budget: ε as a resource

- We don't know which bin is Frank's, so we noise **every** bin. The histogram is 1-sensitive, so adjacent D, D' still differ in only *one* bin: the whole histogram is ε-**DP**.

The privacy budget: ε as a resource

- We don't know which bin is Frank's, so we noise **every** bin. The histogram is 1-sensitive, so adjacent D, D' still differ in only *one* bin: the whole histogram is ε**-DP**.
- Real workloads compute *several* statistics. If M_1 is ε_1-DP and M_2 is ε_2-DP, then $M_1; M_2$ is $(\varepsilon_1+\varepsilon_2)$**-DP** (sequential composition).

The privacy budget: ε as a resource

- We don't know which bin is Frank's, so we noise **every** bin. The histogram is 1-sensitive, so adjacent D, D' still differ in only *one* bin: the whole histogram is ε**-DP**.
- Real workloads compute *several* statistics. If M_1 is ε_1-DP and M_2 is ε_2-DP, then $M_1; M_2$ is $(\varepsilon_1 + \varepsilon_2)$**-DP** (sequential composition).
- **Goal of a DP logic:** given a global budget ε, verify that the released distributions on adjacent D, D' stay within e^{ε}.

The privacy budget: ε as a resource

- We don't know which bin is Frank's, so we noise **every** bin. The histogram is 1-sensitive, so adjacent D, D' still differ in only *one* bin: the whole histogram is ε**-DP**.
- Real workloads compute *several* statistics. If M_1 is ε_1-DP and M_2 is ε_2-DP, then $M_1; M_2$ is $(\varepsilon_1+\varepsilon_2)$**-DP** (sequential composition).
- **Goal of a DP logic:** given a global budget ε, verify that the released distributions on adjacent D, D' stay within e^{ε}.
- Reason **step by step, compositionally**: instead of comparing two distributions of a whole Python or C++ program.

Real DP code: higher-order, stateful combinators

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

$\sim$ *OpenDP, Google DP, IBM diffprivlib.*

- Industrial DP code is built out of **higher-order combinators**.

Real DP code: higher-order, stateful combinators

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

$\sim$ *OpenDP, Google DP, IBM diffprivlib.*

- Industrial DP code is built out of **higher-order combinators**.
- The filter does the **budget bookkeeping**: client passes a cost ε + mechanism M; filter decides whether to run.

Real DP code: higher-order, stateful combinators

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

$\sim$ *OpenDP, Google DP, IBM diffprivlib.*

- Industrial DP code is built out of **higher-order combinators**.
- The filter does the **budget bookkeeping**: client passes a cost ε + mechanism M; filter decides whether to run.
- Needs **local state** (`budget_left`) and a **dynamic check**.

Real DP code: higher-order, stateful combinators

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

∼ *OpenDP, Google DP, IBM diffprivlib.*

- Industrial DP code is built out of **higher-order combinators**.
- The filter does the **budget bookkeeping**: client passes a cost ε + mechanism M; filter decides whether to run.
- Needs **local state** (`budget_left`) and a **dynamic check**.
- Static type systems can't sum runtime costs; existing Hoare logics can't talk about HO mechanisms.

Real DP code: higher-order, stateful combinators

```
class PrivacyFilter:
  def __init__(self, budget):
    self.budget_left = budget

  def try_run(self, cost, M):
    if self.budget_left < cost:
      return None
    self.budget_left -= cost
    return M()
```

∼ *OpenDP, Google DP, IBM diffprivlib.*

- Industrial DP code is built out of **higher-order combinators**.
- The filter does the **budget bookkeeping**: client passes a cost ε + mechanism M; filter decides whether to run.
- Needs **local state** (`budget_left`) and a **dynamic check**.
- Static type systems can't sum runtime costs; existing Hoare logics can't talk about HO mechanisms.

Goal: verify filter + clients in one logic.

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.
- Verify the **distance between output distributions** of ML-like programs over discrete distributions: HO state, recursion, references, the discrete Laplacian, …

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.
- Verify the **distance between output distributions** of ML-like programs over discrete distributions: HO state, recursion, references, the discrete Laplacian, …
- The privacy budget is a **separation logic resource**:
  $↯^{\times}(\varepsilon)$ (multiplicative), $↯^{+}(\delta)$ (additive).

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.
- Verify the **distance between output distributions** of ML-like programs over discrete distributions: HO state, recursion, references, the discrete Laplacian, …
- The privacy budget is a **separation logic resource**:
  $\lightning^{\times}(\varepsilon)$ (multiplicative), $\lightning^{+}(\delta)$ (additive).
- Reason **locally and compositionally** via Hoare quadruples; privacy proofs become program proofs.

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.
- Verify the **distance between output distributions** of ML-like programs over discrete distributions: HO state, recursion, references, the discrete Laplacian, …
- The privacy budget is a **separation logic resource**:
  $\lightning^{\times}(\varepsilon)$ (multiplicative), $\lightning^{+}(\delta)$ (additive).
- Reason **locally and compositionally** via Hoare quadruples; privacy proofs become program proofs.
- **Library of reusable specs:** Laplace, Above Threshold, Sparse Vector, Report Noisy Max, Privacy Filter, Memoization Cache.

Our contribution: Clutch-DP

- A **higher-order separation logic** for differential privacy, built on top of the Iris separation logic framework.
- Verify the **distance between output distributions** of ML-like programs over discrete distributions: HO state, recursion, references, the discrete Laplacian, …
- The privacy budget is a **separation logic resource**:
  $\lightning^{\times}(\varepsilon)$ (multiplicative), $\lightning^{+}(\delta)$ (additive).
- Reason **locally and compositionally** via Hoare quadruples; privacy proofs become program proofs.
- **Library of reusable specs:** Laplace, Above Threshold, Sparse Vector, Report Noisy Max, Privacy Filter, Memoization Cache.
- **Fully mechanised** in the Rocq Prover.

The judgement: relational Hoare quadruples + credits

$$\{P\}\ e_1 \precsim e_2\ \{v_1, v_2.\ Q\}$$

- Pre/post over a **pair** of programs (typically same code, adjacent inputs).

The judgement: relational Hoare quadruples + credits

$$\{P\}\ e_1 \precsim e_2\ \{v_1, v_2.\ Q\}$$

- Pre/post over a **pair** of programs (typically same code, adjacent inputs).
- Full Iris toolbox: $*$, $-\!\!*$, invariants $\boxed{\cdot}$, ghost state, left/right points-to $\ell \mapsto v$ and $\ell \mapsto_s v$.

The judgement: relational Hoare quadruples + credits

$$\{P\}\ e_1 \precsim e_2\ \{v_1, v_2.\ Q\}$$

- Pre/post over a **pair** of programs (typically same code, adjacent inputs).
- Full Iris toolbox: $*$, $-\!\!*$, invariants $\boxed{\cdot}$, ghost state, left/right points-to $\ell \mapsto v$ and $\ell \mapsto_s v$.
- The privacy budget lives in P: $\lightning^{\times}(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning^{\times}(\varepsilon_1) * \lightning^{\times}(\varepsilon_2)$.

The judgement: relational Hoare quadruples + credits

$$\{P\}\ e_1 \precsim e_2\ \{v_1, v_2.\ Q\}$$

- Pre/post over a **pair** of programs (typically same code, adjacent inputs).
- Full Iris toolbox: $*$, $-\!\!*$, invariants $\boxed{\cdot}$, ghost state, left/right points-to $\ell \mapsto v$ and $\ell \mapsto_s v$.
- The privacy budget lives in P: $\lightning^{\times}(\varepsilon_1 + \varepsilon_2) \dashv\vdash \lightning^{\times}(\varepsilon_1) * \lightning^{\times}(\varepsilon_2)$.

Familiar deterministic rules "just work"

$$\frac{\{P * \ell \mapsto w\}\ ()\ \precsim e'\ \{Q\}}{\{P * \ell \mapsto v\}\ \ell \leftarrow w\ \precsim e'\ \{Q\}}\ \textsc{Store-l}$$

Internal DP and adequacy

Internal DP
M is *internally* (ε, δ)-DP if

a predicate in the logic

$$\forall x \sim y.\ \left\{ \lightning^{\times}(\varepsilon) \ * \ \lightning^{+}(\delta) \right\} M\,x \precsim M\,y\,\{v, v'.\, v = v'\}.$$

Adequacy theorem
Internal (ε, δ)-DP $\implies$ external (ε, δ)-DP:

ties the logic to plain DP

$$\forall x \sim y, S.\ \ \Pr[\mathsf{exec}(M\,x) \in S] \ \leq \ e^{\varepsilon} \cdot \Pr[\mathsf{exec}(M\,y) \in S] \ + \ \delta.$$

- Work internally: use the whole logic: invariants, ghost state, recursion.

Internal DP and adequacy

Internal DP
M is *internally* (ε, δ)-DP if

a predicate in the logic

$$\forall x \sim y.\ \left\{ \lightning^{\times}(\varepsilon) \ * \ \lightning^{+}(\delta) \right\} M\,x \precsim M\,y\,\{v, v'.\, v = v'\}.$$

Adequacy theorem
Internal (ε, δ)-DP $\implies$ external (ε, δ)-DP:

ties the logic to plain DP

$$\forall x \sim y, S.\ \ \Pr[\mathsf{exec}(M\,x) \in S] \ \leq \ e^{\varepsilon} \cdot \Pr[\mathsf{exec}(M\,y) \in S] \ + \ \delta.$$

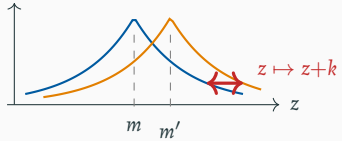
- Work internally: use the whole logic: invariants, ghost state, recursion.
- **Pay credits to "buy equality"** in the postcondition, reason about *values*, not distributions.

Sampling rule: Laplace-shift

Laplace-shift
If $|k + m - m'| \leq c$ then

$$\left\{ \lightning^{\times}(c \cdot \varepsilon) \right\} \ \text{Laplace}\ \varepsilon\, m \precsim \text{Laplace}\ \varepsilon\, m'\ \{z, z' . z' = z + k\}.$$

Read the rule as *two separate cases*, both paying $\lightning^{\times}(c\,\varepsilon)$:



cost: $\lightning^{\times}(c\,\varepsilon)$

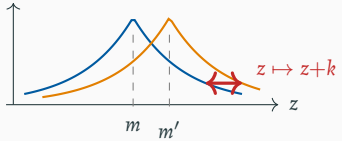
Sampling rule: Laplace-shift

Laplace-shift
If $|k + m - m'| \leq c$ then

$$\left\{\lightning^{\times}(c \cdot \varepsilon)\right\} \ \text{Laplace}\ \varepsilon\, m \precsim \text{Laplace}\ \varepsilon\, m'\ \{z, z'.\, z' = z + k\}.$$

Read the rule as *two separate cases*, both paying $\lightning^{\times}(c\,\varepsilon)$:

- **Case (i)** $k{=}0$, $|m{-}m'| \leq c$: pay $\lightning^{\times}(c\,\varepsilon)$ to absorb the *input distance*; get $z = z'$.



cost: $\lightning^{\times}(c\,\varepsilon)$

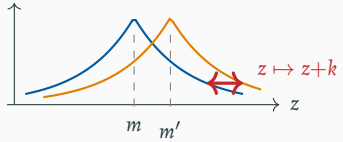
Sampling rule: Laplace-shift

Laplace-shift
If $|k + m - m'| \leq c$ then

$$\left\{\lightning^{\times}(c \cdot \varepsilon)\right\} \ \text{Laplace}\ \varepsilon\, m \precsim \text{Laplace}\ \varepsilon\, m'\ \{z, z'.\, z' = z + k\}.$$

Read the rule as *two separate cases*, both paying $\lightning^{\times}(c\,\varepsilon)$:

- **Case (i)** $k{=}0$, $|m{-}m'| \leq c$: pay $\lightning^{\times}(c\,\varepsilon)$ to absorb the *input distance*; get $z = z'$.
- **Case (ii)** $m{=}m'$, $|k| \leq c$: pay $\lightning^{\times}(c\,\varepsilon)$ to *separate the outputs* by k; get $z' = z + k$.



cost: $\lightning^{\times}(c\,\varepsilon)$

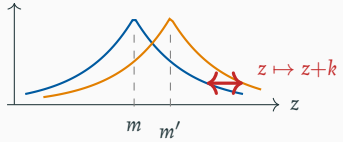
Sampling rule: Laplace-shift

Laplace-shift
If $|k + m - m'| \le c$ then

$$\left\{\lightning^{\times}(c \cdot \varepsilon)\right\} \ \text{Laplace}\ \varepsilon\, m \precsim \text{Laplace}\ \varepsilon\, m'\ \{z, z'.\, z' = z + k\}.$$

Read the rule as *two separate cases*, both paying $\lightning^{\times}(c\,\varepsilon)$:

- **Case (i)** $k{=}0$, $|m{-}m'| \le c$: pay $\lightning^{\times}(c\,\varepsilon)$ to absorb the *input distance*; get $z = z'$.
- **Case (ii)** $m{=}m'$, $|k| \le c$: pay $\lightning^{\times}(c\,\varepsilon)$ to *separate the outputs* by k; get $z' = z + k$.



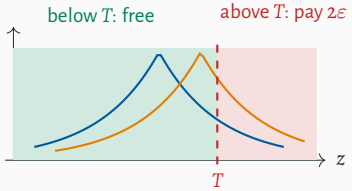
cost: $\lightning^{\times}(c\,\varepsilon)$

Case (i) with $c{=}1$ recovers **Laplace is ε-DP**.

Sampling rule: Laplace-choice

Laplace-choice
For $|m - m'| \leq 1$ and $T \in \mathbb{Z}$:

$$\left\{\lightning^{\times}(2\varepsilon)\right\}\ \text{Laplace}\ \varepsilon\ m \precsim \text{Laplace}\ \varepsilon\ m' \left\{ \begin{array}{l} z, z'.\ (T \leq z \wedge T{+}1 \leq z') \vee \\ \qquad (z < T \wedge z' < T{+}1 * \lightning^{\times}(2\varepsilon)) \end{array} \right\}.$$

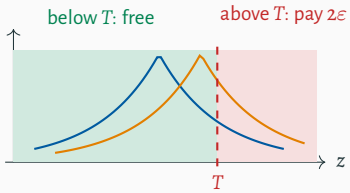


- Outcomes partitioned by threshold T.

Sampling rule: Laplace-choice

Laplace-choice
For $|m - m'| \leq 1$ and $T \in \mathbb{Z}$:

$$\left\{\lightning^{\times}(2\varepsilon)\right\}\ \text{Laplace}\ \varepsilon\ m \precsim \text{Laplace}\ \varepsilon\ m' \left\{ \begin{array}{l} z, z'.\ (T \leq z \wedge T{+}1 \leq z') \vee \\ \qquad (z < T \wedge z' < T{+}1 * \lightning^{\times}(2\varepsilon)) \end{array} \right\}.$$

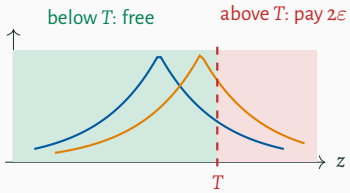


- Outcomes partitioned by threshold T.
- **Above** T: pay $\lightning^{\times}(2\varepsilon)$.

Sampling rule: Laplace-choice

Laplace-choice
For $|m - m'| \leq 1$ and $T \in \mathbb{Z}$:

$$\left\{\lightning^{\times}(2\varepsilon)\right\} \text{ Laplace } \varepsilon\, m \precsim \text{Laplace } \varepsilon\, m' \left\{ \begin{array}{l} z, z'.\ (T \leq z \wedge T{+}1 \leq z') \vee \\ \qquad (z < T \wedge z' < T{+}1 * \lightning^{\times}(2\varepsilon)) \end{array} \right\}.$$

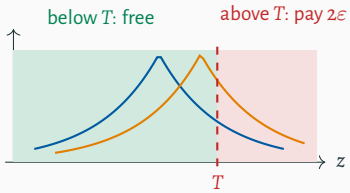


- Outcomes partitioned by threshold T.
- **Above** T: pay $\lightning^{\times}(2\varepsilon)$.
- **Below** T: no leak, *recover* $\lightning^{\times}(2\varepsilon)$.

Sampling rule: Laplace-choice

Laplace-choice
For $|m - m'| \leq 1$ and $T \in \mathbb{Z}$:

$$\left\{ \lightning^{\times}(2\varepsilon) \right\} \ \text{Laplace}\ \varepsilon\, m \precsim \text{Laplace}\ \varepsilon\, m' \left\{ \begin{array}{l} z, z'.\ (T \leq z \wedge T{+}1 \leq z') \vee \\ \qquad (z < T \wedge z' < T{+}1 * \lightning^{\times}(2\varepsilon)) \end{array} \right\}.$$



- Outcomes partitioned by threshold T.
- **Above** T: pay $\lightning^{\times}(2\varepsilon)$.
- **Below** T: no leak, *recover* $\lightning^{\times}(2\varepsilon)$.

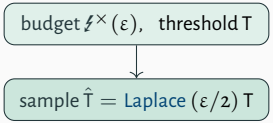
"Only pay when you reveal".

Above Threshold, operationally

budget $\lightning^{\times}(\varepsilon)$, threshold T

Start with a privacy budget $\lightning^{\times}(\varepsilon)$ and a threshold T.

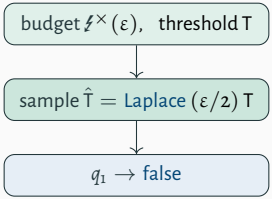
Above Threshold, operationally



Start with a privacy budget $\lightning^{\times}(\varepsilon)$ and a threshold T.

Sample the noisy threshold $\hat{T}$; spends part of the budget.

Above Threshold, operationally

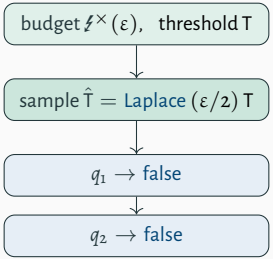


Start with a privacy budget $\lightning^{\times}(\varepsilon)$ and a threshold T.

Sample the noisy threshold $\hat{T}$; spends part of the budget.

q_1: noisy result below $\hat{T}$; budget intact.

Above Threshold, operationally



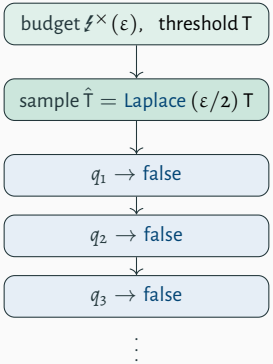
Start with a privacy budget $\lightning^{\times}(\varepsilon)$ and a threshold T.

Sample the noisy threshold $\hat{T}$; spends part of the budget.

q_1: noisy result below $\hat{T}$; budget intact.

q_2: noisy result below $\hat{T}$; budget intact.

Above Threshold, operationally



Start with a privacy budget $\lightning^{\times}(\varepsilon)$ and a threshold T.

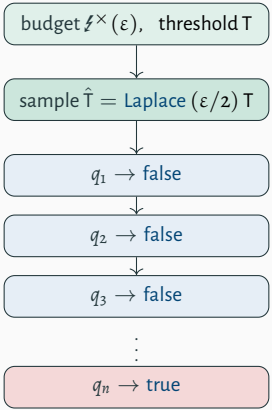
Sample the noisy threshold $\hat{T}$; spends part of the budget.

q_1: noisy result below $\hat{T}$; budget intact.

q_2: noisy result below $\hat{T}$; budget intact.

q_3: noisy result below $\hat{T}$; budget intact.

Above Threshold, operationally



Start with a privacy budget $\not{\varepsilon}^{\times}(\varepsilon)$ and a threshold T.

Sample the noisy threshold $\hat{T}$; spends part of the budget.

q_1: noisy result below $\hat{T}$; budget intact.

q_2: noisy result below $\hat{T}$; budget intact.

q_3: noisy result below $\hat{T}$; budget intact.

Eventually a query exceeds $\hat{T}$; **budget consumed**, the closure stops here.

Above Threshold: code and spec

Code

```
let above_threshold ε T =
  let T̂ = Laplace (ε/2) T in
  let f q db =
    let x = q db in
    let y = Laplace (ε/4) x in
    T̂ ≤ y
  in f
```

$\{ \lightning^{\times}(\varepsilon) \}$

above_threshold ε T $\precsim$ above_threshold ε T

$\{ f, f'.\ \exists AUTH.\ AUTH * \quad \}$

Specification

- Pay $\lightning^{\times}(\varepsilon)$ up front; receive closure f + token $AUTH$.

Above Threshold: code and spec

Code

```
let above_threshold ε T =
  let T̂ = Laplace (ε/2) T in
  let f q db =
    let x = q db in
    let y = Laplace (ε/4) x in
    T̂ ≤ y
  in f
```

Specification

- Pay $\lightning^{\times}(\varepsilon)$ up front; receive closure f + token $AUTH$.
- Each call on 1-sens(q), $\mathrm{db} \sim \mathrm{db}'$ returns the same boolean.

$\{\lightning^{\times}(\varepsilon)\}$
above_threshold ε T $\precsim$ above_threshold ε T
$\{\mathrm{f}, \mathrm{f}'. \exists AUTH.\ AUTH * \forall \mathrm{db} \sim \mathrm{db}', \mathrm{q}.\ \{AUTH * 1\text{-sens}(\mathrm{q})\}\ \mathrm{f}\ \mathrm{q}\ \mathrm{db} \precsim \mathrm{f}'\ \mathrm{q}\ \mathrm{db}'\ \{b\, b'.\ \}\ \}$

Above Threshold: code and spec

Code

```
let above_threshold ε T =
  let T̂ = Laplace (ε/2) T in
  let f q db =
    let x = q db in
    let y = Laplace (ε/4) x in
    T̂ ≤ y
  in f
```

Specification

- Pay $\lightning^{\times}(\varepsilon)$ up front; receive closure f + token $AUTH$.
- Each call on 1-sens(q), $\mathrm{db} \sim \mathrm{db}'$ returns the same boolean.
- $AUTH$ is consumed when b=true, *recovered* when b=false.

$\{\lightning^{\times}(\varepsilon)\}$
above_threshold ε T $\precsim$ above_threshold ε T
$\{\mathrm{f}, \mathrm{f}'. \exists AUTH.\ AUTH * \forall \mathrm{db} \sim \mathrm{db}', \mathrm{q}.\ \{AUTH * 1\text{-sens}(\mathrm{q})\}\ \mathrm{f\ q\ db} \precsim \mathrm{f'\ q\ db'}\ \{b\ b'.\ b{=}b' * (\mathrm{not}\ b \mathrel{-\!\!*} AUTH)\}\ \}$

Above Threshold: code and spec

Code

```
let above_threshold ε T =
  let T̂ = Laplace (ε/2) T in
  let f q db =
    let x = q db in
    let y = Laplace (ε/4) x in
    T̂ ≤ y
  in f
```

Specification

- Pay $\lightning^{\times}(\varepsilon)$ up front; receive closure f + token $AUTH$.
- Each call on 1-sens(q), $\mathsf{db} \sim \mathsf{db}'$ returns the same boolean.
- $AUTH$ is consumed when b=true, *recovered* when b=false.
- **Interactive:** q may depend on prior booleans.

$$\{\lightning^{\times}(\varepsilon)\}\ \text{above\_threshold}\ \varepsilon\ \mathsf{T} \precsim \text{above\_threshold}\ \varepsilon\ \mathsf{T}$$

$$\{\mathsf{f}, \mathsf{f}'.\ \exists AUTH.\ AUTH * \forall \mathsf{db} \sim \mathsf{db}', \mathsf{q}.\ \{AUTH * \text{1-sens}(\mathsf{q})\}\ \mathsf{f}\ \mathsf{q}\ \mathsf{db} \precsim \mathsf{f}'\ \mathsf{q}\ \mathsf{db}'\ \{b\ b'.\ b{=}b' * (\text{not}\ b \mathrel{-\!\!*} AUTH)\}\ \}$$

AT proof sketch

Start with $↯^{\times}(\varepsilon)$; reach the spec in four steps.

1. Split $↯^{\times}(\varepsilon) = ↯^{\times}(\varepsilon/2) \ast ↯^{\times}(\varepsilon/2)$.

AT proof sketch

Start with $↯^{\times}(\varepsilon)$; reach the spec in four steps.

1. Split $↯^{\times}(\varepsilon) = ↯^{\times}(\varepsilon/2) \ast ↯^{\times}(\varepsilon/2)$.
2. Apply `Laplace-shift` on $\hat{\mathsf{T}}, \hat{\mathsf{T}}'$ with $k = 1, c = 1$: cost $↯^{\times}(\varepsilon/2)$, get $\hat{\mathsf{T}}' = \hat{\mathsf{T}} + 1$.

AT proof sketch

Start with $\lightning^{\times}(\varepsilon)$; reach the spec in four steps.

1. Split $\lightning^{\times}(\varepsilon) = \lightning^{\times}(\varepsilon/2) \ast \lightning^{\times}(\varepsilon/2)$.
2. Apply `Laplace-shift` on $\hat{\mathsf{T}}, \hat{\mathsf{T}}'$ with $k = 1, c = 1$: cost $\lightning^{\times}(\varepsilon/2)$, get $\hat{\mathsf{T}}' = \hat{\mathsf{T}} + 1$.
3. Define $AUTH \triangleq \lightning^{\times}(\varepsilon/2)$ (the other half).

AT proof sketch

Start with $\lightning^{\times}(\varepsilon)$; reach the spec in four steps.

1. Split $\lightning^{\times}(\varepsilon) = \lightning^{\times}(\varepsilon/2) \ast \lightning^{\times}(\varepsilon/2)$.
2. Apply `Laplace-shift` on $\hat{T}, \hat{T}'$ with $k = 1, c = 1$: cost $\lightning^{\times}(\varepsilon/2)$, get $\hat{T}' = \hat{T} + 1$.
3. Define $AUTH \triangleq \lightning^{\times}(\varepsilon/2)$ (the other half).
4. Per query, 1-sens(q) and adjacency give $|x - x'| \leq 1$, so apply `Laplace-choice` on (y, y') at threshold $\hat{T}$:
   - $y \geq \hat{T}$: both return true; $AUTH$ consumed.
   - $y < \hat{T}$: both return false; $AUTH$ recovered.

AT proof sketch

Start with $\lightning^{\times}(\varepsilon)$; reach the spec in four steps.

1. Split $\lightning^{\times}(\varepsilon) = \lightning^{\times}(\varepsilon/2) \ast \lightning^{\times}(\varepsilon/2)$.
2. Apply `Laplace-shift` on $\hat{T}, \hat{T}'$ with $k = 1, c = 1$: cost $\lightning^{\times}(\varepsilon/2)$, get $\hat{T}' = \hat{T} + 1$.
3. Define $AUTH \triangleq \lightning^{\times}(\varepsilon/2)$ (the other half).
4. Per query, 1-sens(q) and adjacency give $|x - x'| \leq 1$, so apply `Laplace-choice` on (y, y') at threshold $\hat{T}$:
   - $y \geq \hat{T}$: both return true; $AUTH$ consumed.
   - $y < \hat{T}$: both return false; $AUTH$ recovered.

All four steps are direct applications of Clutch-DP rules. □

In the paper

Applications

- **Combinators:** Privacy Filter, Cache.
- **Mechanisms:** SVT, Report Noisy Max.
- **Clients:** auto_avg, adaptive counting, streaming SVT.

How the logic is built

- Approx. (ε, δ)-couplings, compositional step-by-step WP.
- New *choice-composition* theorem: powers `Laplace-choice`.

Summary Clutch-DP = HO sep-logic for DP, credits $↯^{\times}(\varepsilon) * ↯^{+}(\delta)$, `Laplace-choice`, modular library, fully mechanised in Rocq.

Thank you!

`clutch-project.org` `pgh@cs.au.dk`

Encore

BACKUP SLIDES