SampCert

Foundations for Verified Differential Privacy

Markus de Medeiros
Joseph Tassarotti

Muhammad Naveed
Tancrède Lepoint
Temesghen Kahsai
Tristan Ravitch
Anjali Joshi
Aws Albarghouthi
Jean-Baptiste Tristan

Aws Albarghouthi

Privacy is tricky

Privacy is tricky

Oversimplified privacy models

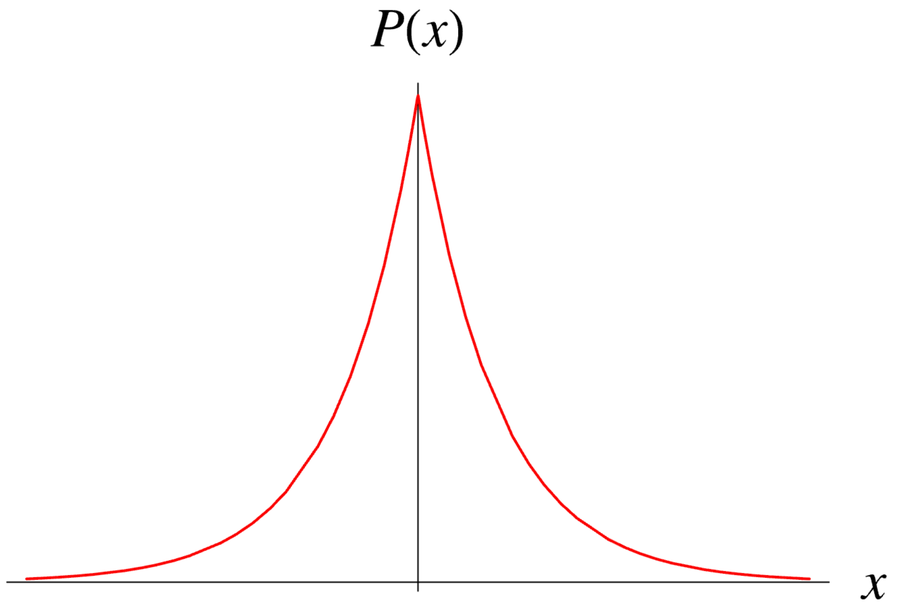
Precision leaks Mironov (2012), Tumult Labs (**2021**)

Sensitivity errors (overflow) Casacuberta et al. (**2022**)

Privacy is tricky

Oversimplified privacy models

On paper: $\epsilon = 10^{-6}$



mathworld.wolfram.com

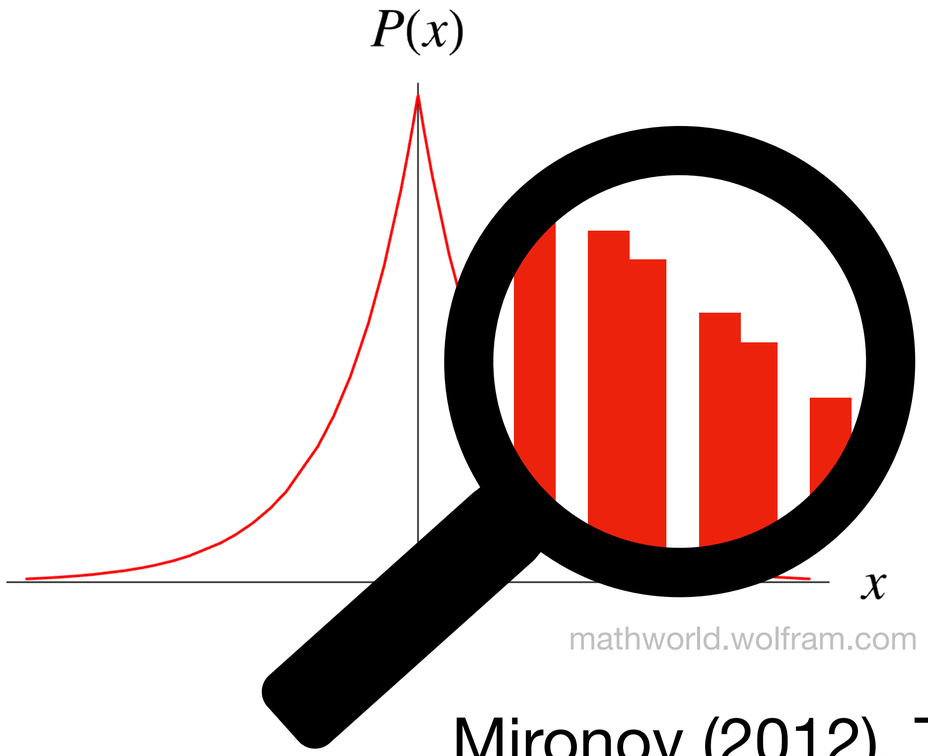
Precision leaks Mironov (2012), Tumult Labs (**2021**)

Sensitivity errors (overflow) Casacuberta et al. (**2022**)

Privacy is tricky

Oversimplified privacy models

On paper: $\epsilon = 10^{-6}$



In code: 40%

Precision leaks

Sensitivity errors (overflow)

Mironov (2012), Tumult Labs (**2021**)

Casacuberta et al. (**2022**)

Privacy is tricky

Oversimplified privacy models

Buggy implementations

Algorithmic errors Lyu et al. (2017)

Budget errors *Cebere et al. (**2026**)*

Privacy is tricky

Oversimplified privacy models

Buggy implementations

Many subtly wrong implementations of textbook SVT

Algorithmic errors Lyu et al. (2017)

Budget errors *Cebere et al. (**2026**)*

Privacy is tricky

Oversimplified privacy models

Buggy implementations

Incomplete attacker models

Timing leaks *Jin et al. (2021)*

Question:

When we say code is differentially private,

how can we check that it actually is?

Question:

When we say code is differentially private,

how can we check that it actually is?

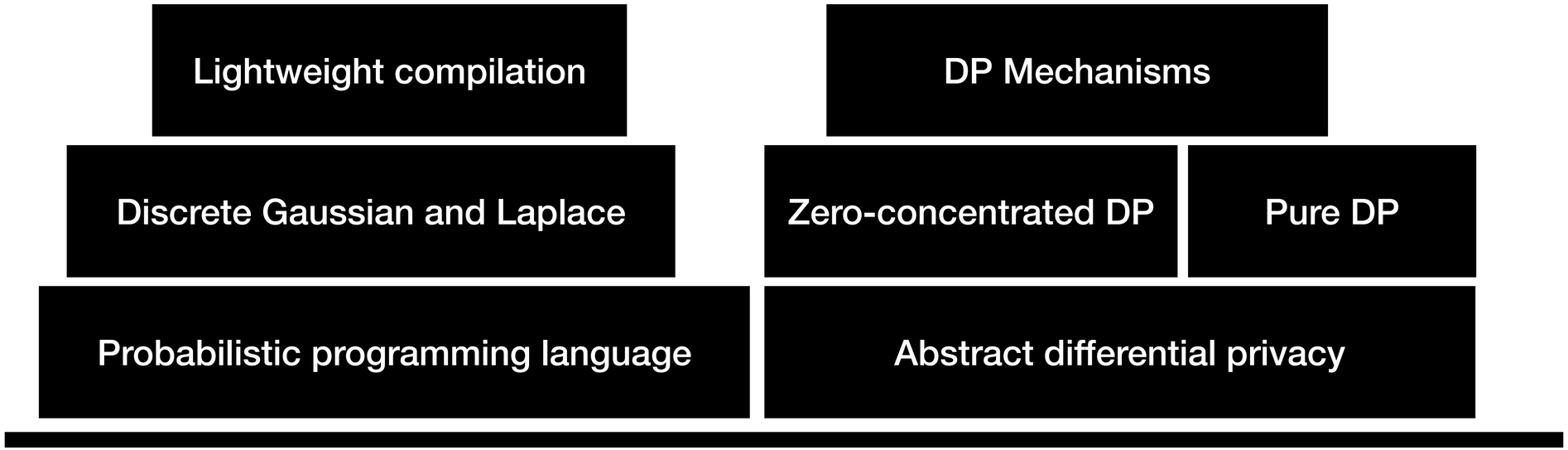
Answer:

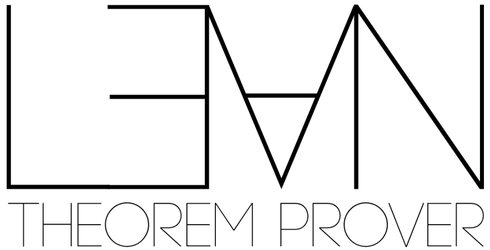
Formal Verification

SampCert

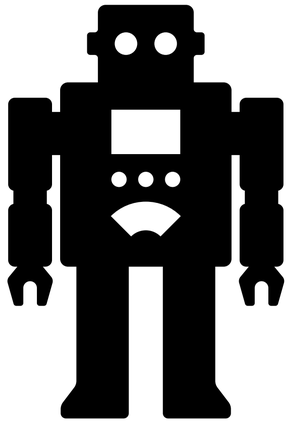
SampCert

Foundations for Verified Differential Privacy





Privacy Engineer (you)



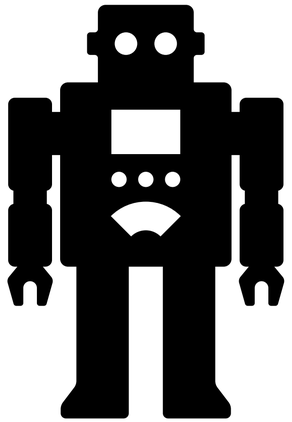
Proof Checker

Code

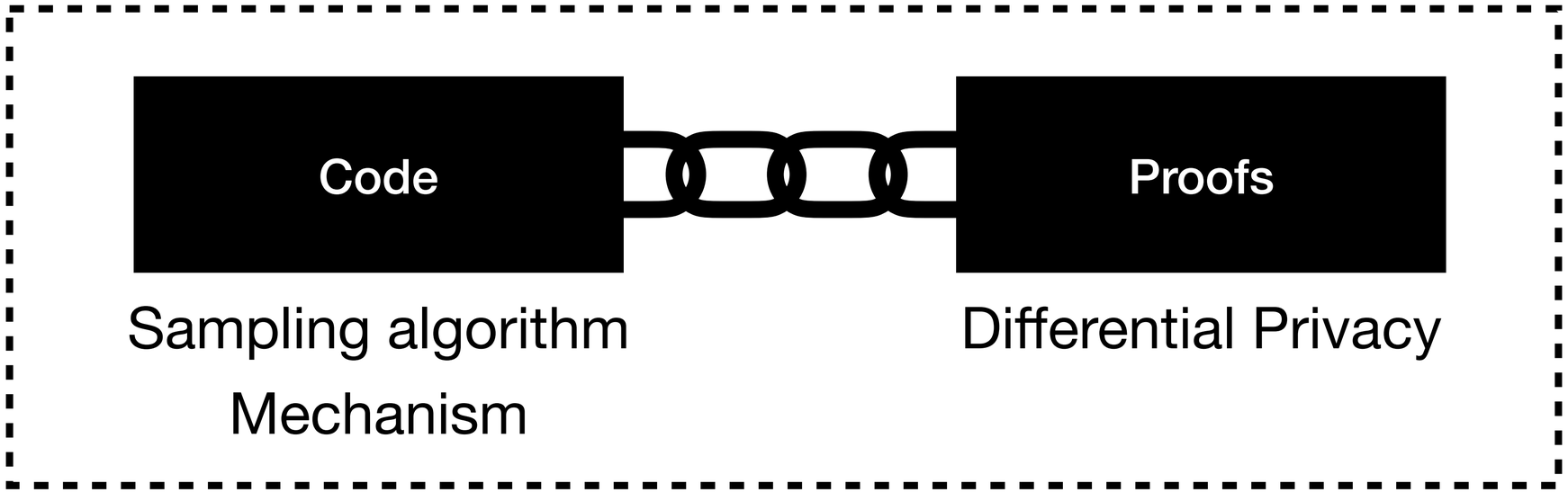
Sampling algorithm

Mechanism

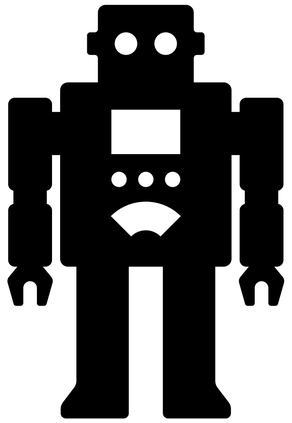
Privacy Engineer (you)



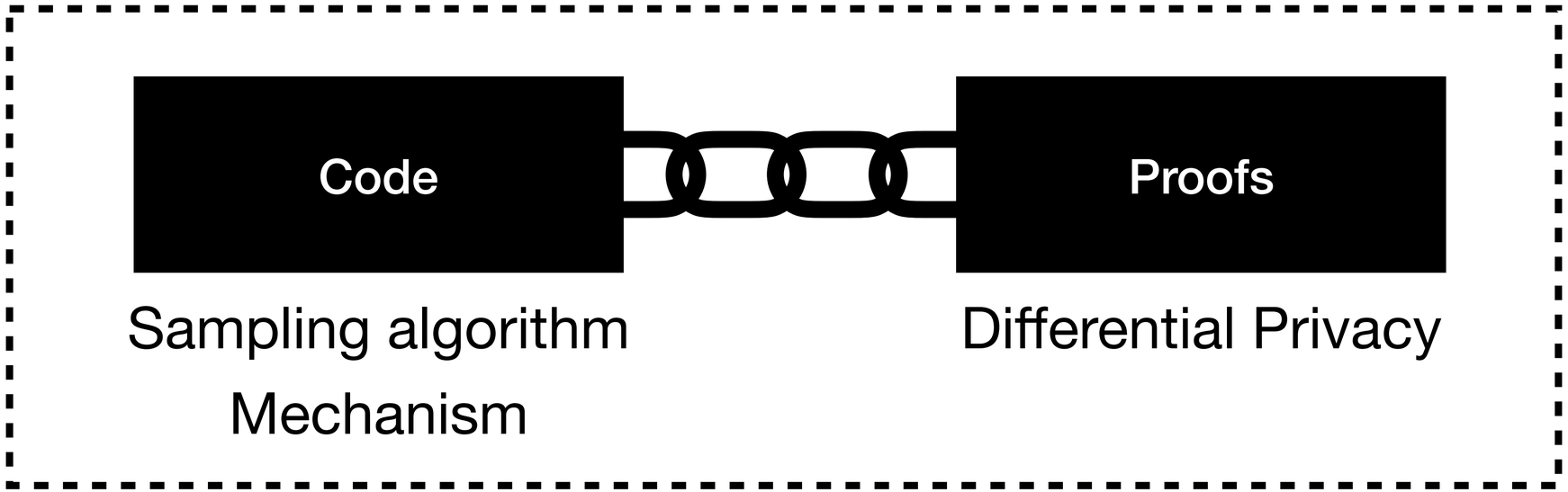
Proof Checker



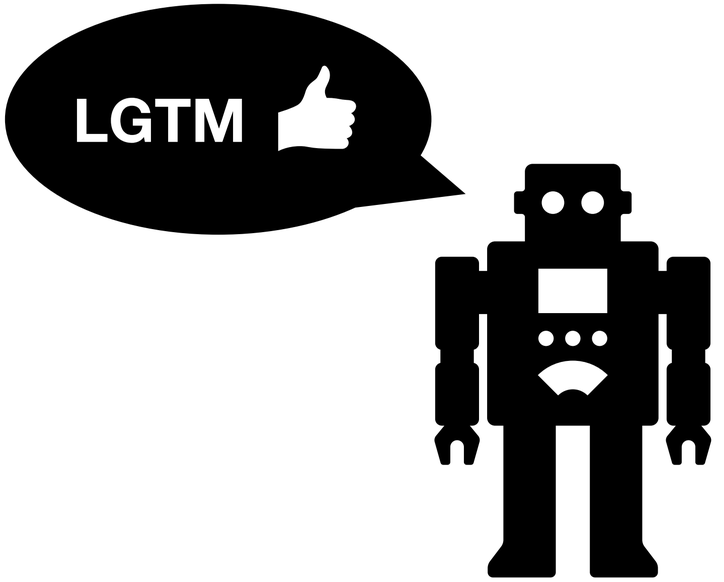
Privacy Engineer (you)



Proof Checker

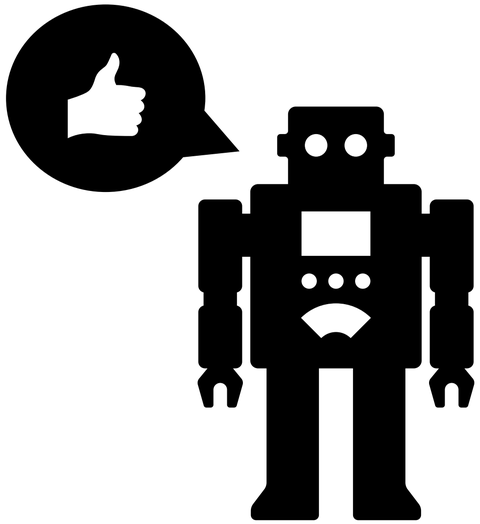


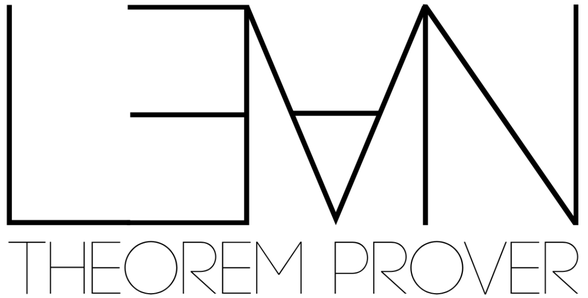
Privacy Engineer (you)



Proof Checker

What does formal methods get you?





✓ Your privacy proof is correct

Compilation of your code to binary

Coding in SampCert

Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}\left(0, \sigma^2\right)$.

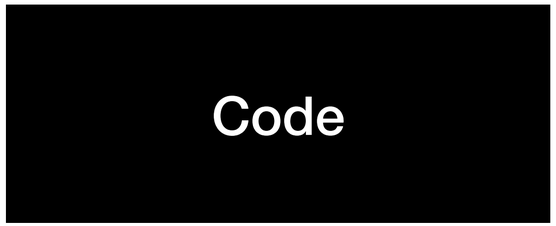
Set $t \leftarrow \lfloor \sigma \rfloor + 1$
loop
  Sample $Y \leftarrow \mathrm{Lap}_{\mathbb{Z}}(t)$
  Sample $C \leftarrow \mathsf{Bernoulli}(\exp(-(|Y| - \sigma^2/t)^2/2\sigma^2))$.
  If $C = 0$, reject and restart.
  If $C = 1$, return Y as output.

```
/-- Sample a candidate for the Discrete Gaussian with variance ``num/den``. -/
def DiscreteGaussianSampleLoop (num den t : PNat) (mix : ℕ) : SLang (Int × Bool) := do
  let Y : Int ← DiscreteLaplaceSampleMixed t 1 mix
  let y : Nat := Int.natAbs Y
  let n : Nat := (Int.natAbs (Int.sub (y * t * den) num))^2
  let d : PNat := 2 * num * t^2 * den
  let C ← BernoulliExpNegSample n d
  return (Y,C)

/-- Sample a value from the Discrete Gaussian with variance ``(num/den)``^2. -/
def DiscreteGaussianSample (num : PNat) (den : PNat) (mix : ℕ) : SLang ℤ := do
  let ti : Nat := num.val / den
  let t : PNat := ⟨ ti + 1 , by simp only [add_pos_iff, zero_lt_one, or_true] ⟩
  let num := num^2
  let den := den^2
  let r ← probUntil (DiscreteGaussianSampleLoop num den t mix) (λ x : Int × Bool ⇒ x.2)
  return r.1
```

Canone, Kamath, Steinke. *The discrete Gaussian for differential privacy.*

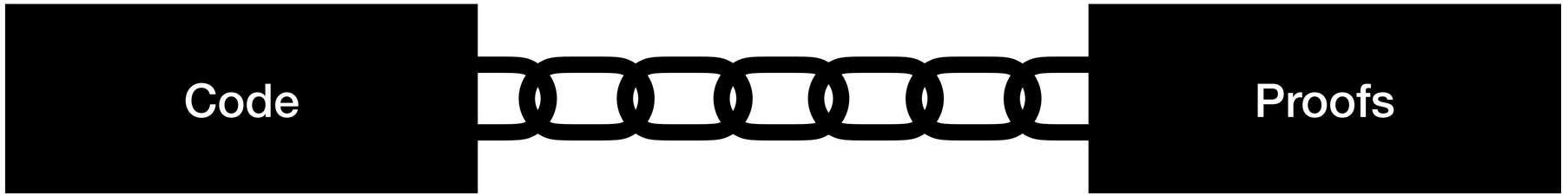
Coding in SampCert



Minimal language with:

- Pure operations
- Uniform byte samples
- Loops and recursion

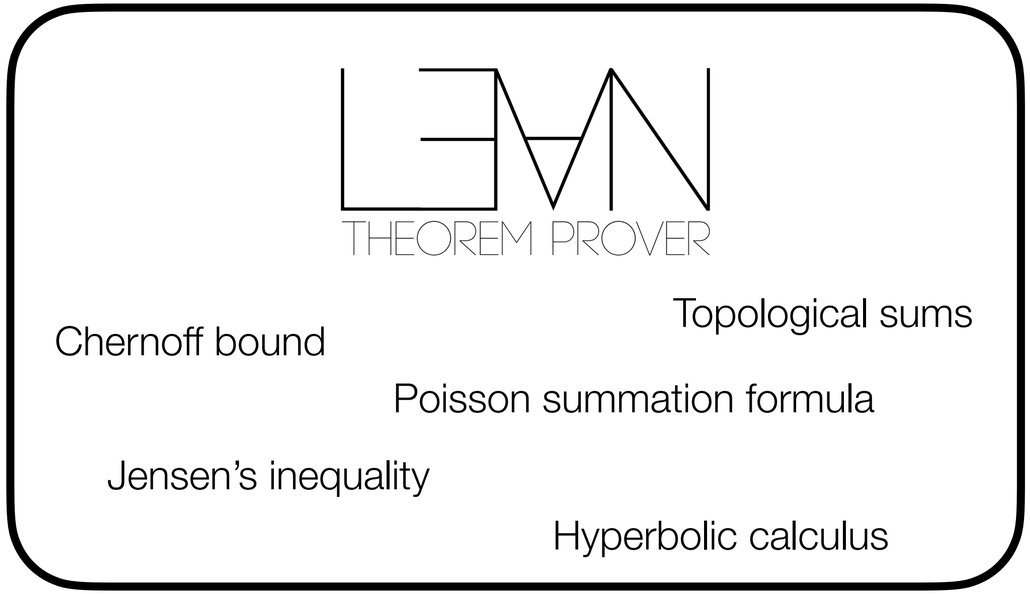
Coding in SampCert



Minimal language with:

- Pure operations
- Uniform byte samples
- Loops and recursion

Probability Mass Functions



Proving correctness in SampCert

```
theorem DiscreteGaussianSample_apply (num den : Positive) : ∀ z : ℤ,
    DiscreteGaussianSample num den z = e ^ ((-(z - μ)^2) / (2 * σ^2)) / (Norm num den)
```

Proving correctness in SampCert

```
theorem DiscreteGaussianSample_apply (num den : Positive) : ∀ z : ℤ,
    DiscreteGaussianSample num den z = e ^ ((-(z - μ)^2) / (2 * σ^2)) / (Norm num den)
```

Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}\left(0, \sigma^2\right)$.
  Set $t \leftarrow \lfloor \sigma \rfloor + 1$
  loop
    Sample $Y \leftarrow \text{Lap}_{\mathbb{Z}}(t)$
    Sample $C \leftarrow \text{Bernoulli}(\exp(-(|Y| - \sigma^2/t)^2/2\sigma^2))$.
    If $C = 0$, reject and restart.
    If $C = 1$, return Y as output.

Proving correctness in SampCert

```
theorem DiscreteGaussianSample_apply (num den : Positive) : ∀ z : ℤ,
    DiscreteGaussianSample num den z = e ^ ((-(z - μ)^2) / (2 * σ^2)) / (Norm num den)
```

Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$.
  Set $t \leftarrow \lfloor \sigma \rfloor + 1$
  loop
    Sample $Y \leftarrow \text{Lap}_{\mathbb{Z}}(t)$
    Sample $C \leftarrow \text{Bernoulli}(\exp(-(|Y| - \sigma^2/t)^2/2\sigma^2))$.
    If $C = 0$, reject and restart.
    If $C = 1$, return Y as output.

$$\frac{e^{-(z-\mu)^2/2\sigma^2}}{\sum_{z \in \mathbb{Z}} e^{-(z-\mu)^2/2\sigma^2}}$$

Proving correctness in SampCert

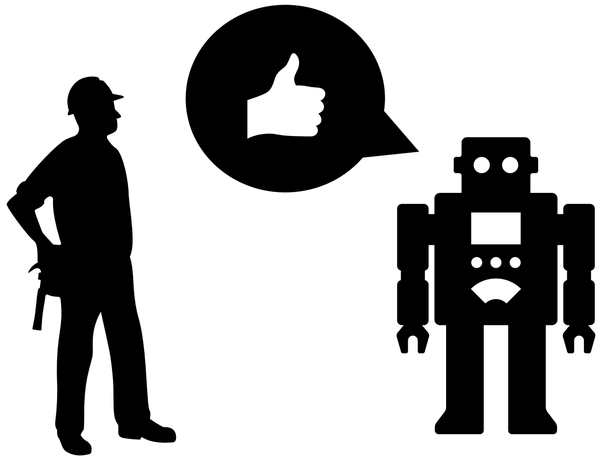
```
theorem DiscreteGaussianSample_apply (num den : Positive) : ∀ z : ℤ,
    DiscreteGaussianSample num den z = e ^ ((-(z - μ)^2) / (2 * σ^2)) / (Norm num den)
```

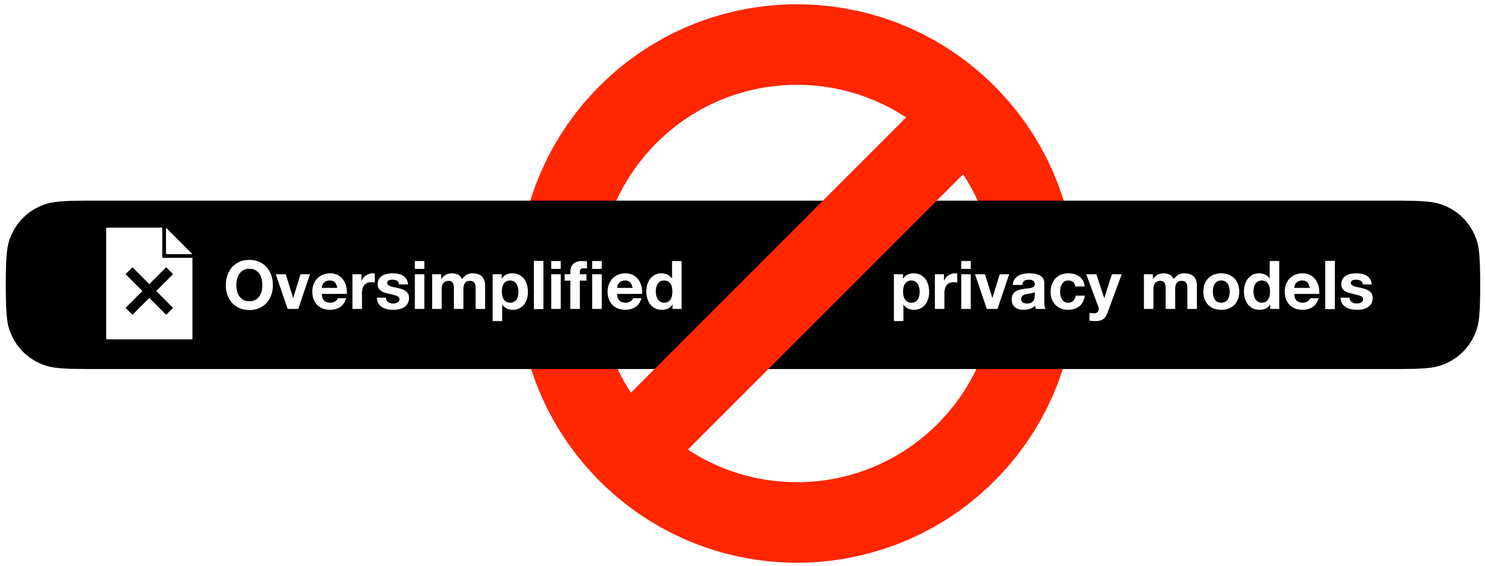
Algorithm 3 Algorithm for Sampling a Discrete Gaussian

Input: Parameter $\sigma^2 > 0$.
Output: One sample from $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$.

Set $t \leftarrow \lfloor \sigma \rfloor + 1$
loop
  Sample $Y \leftarrow \text{Lap}_{\mathbb{Z}}(t)$
  Sample $C \leftarrow \text{Bernoulli}(\exp(-(|Y| - \sigma^2/t)^2/2\sigma^2))$.
  If $C = 0$, reject and restart.
  If $C = 1$, return Y as output.

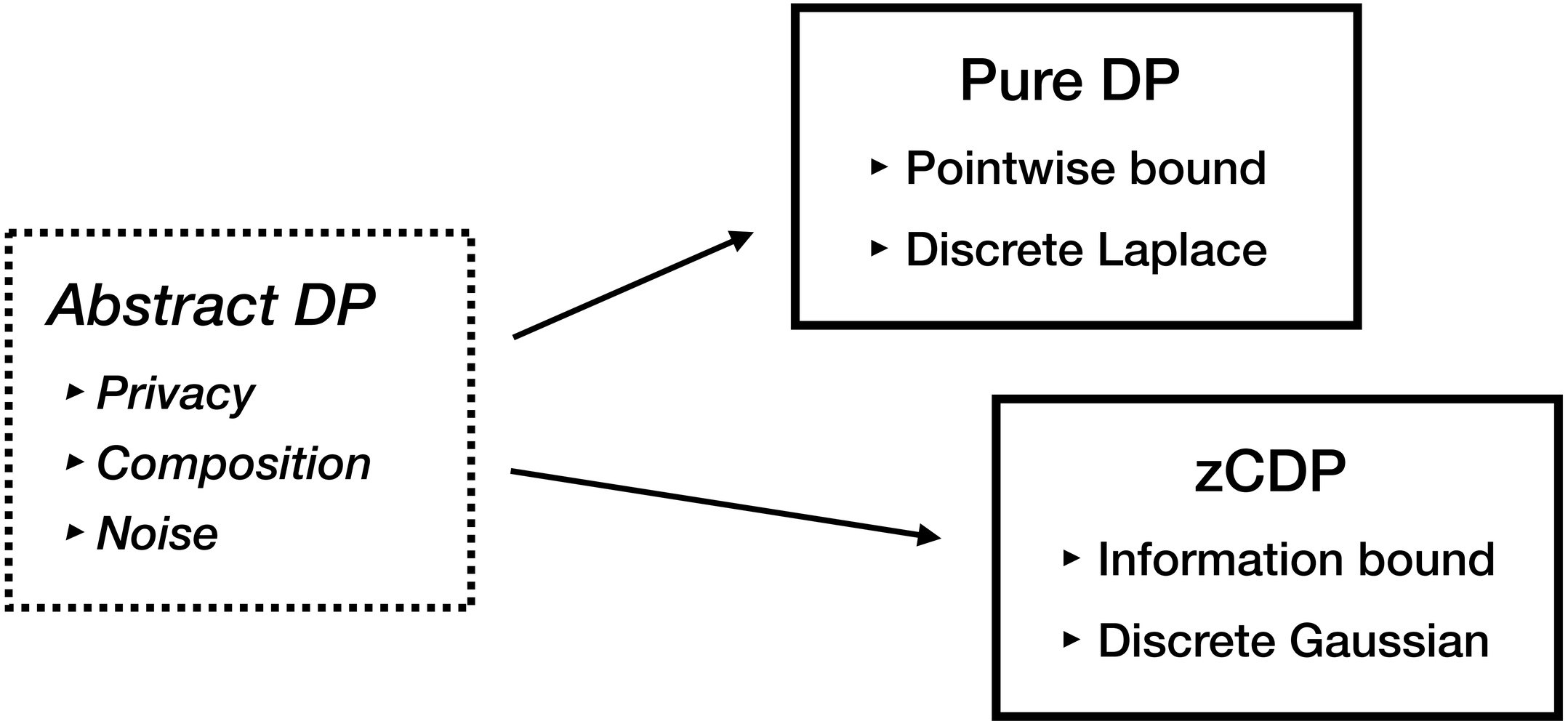
$$\frac{e^{-(z-\mu)^2/2\sigma^2}}{\sum_{z \in \mathbb{Z}} e^{-(z-\mu)^2/2\sigma^2}}$$



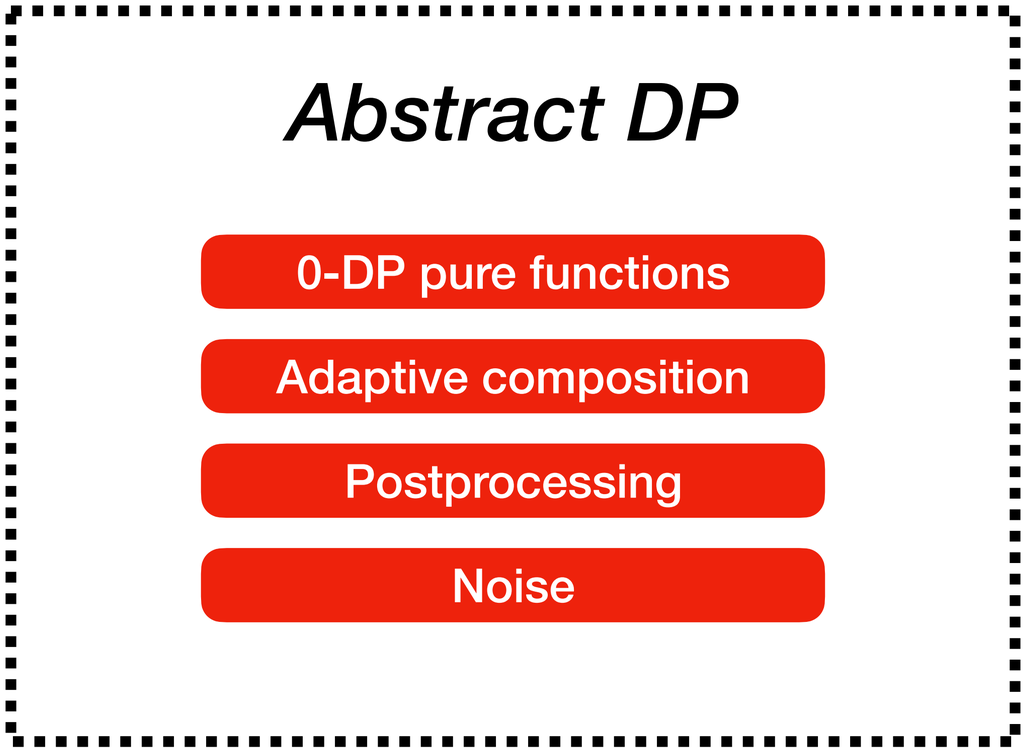


Buggy floating-point samplers fail here

Defining Privacy in SampCert

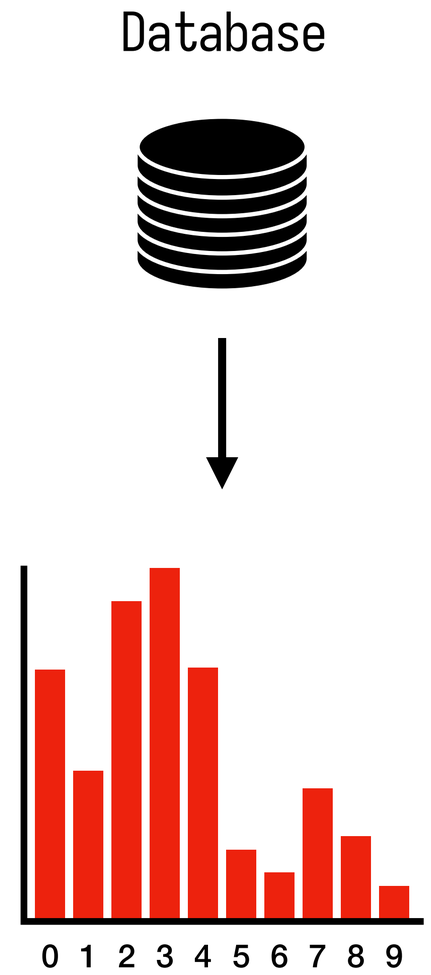


Abstract Differential Privacy



Abstract Privacy Proofs

```
def privHistogram (D : Database) (n : ℕ) := do
  match n with
  | 0 ⇒ return emptyHistogram
  | n' + 1 ⇒ do
     let h ← privHistogram D (n - 1)
     let c ← count D n + noise ε
     updateHistogram h n c
```



Abstract Privacy Proofs

```
def privHistogram (D : Database) (n : ℕ) := do
  match n with
  | 0 ⇒ return emptyHistogram
  | n' + 1 ⇒ do
    let h ← privHistogram D (n - 1)
    let c ← count D n + noise ε
    updateHistogram h n c
```

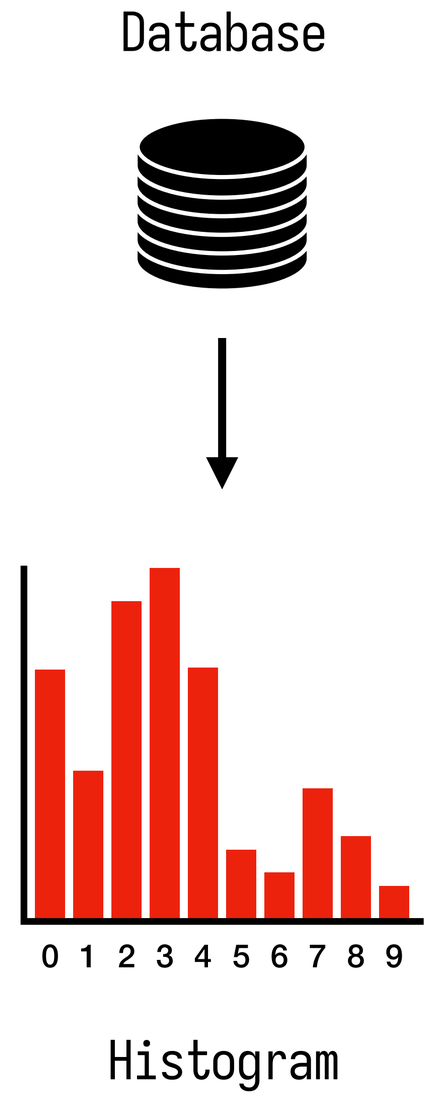
pure

composition

composition

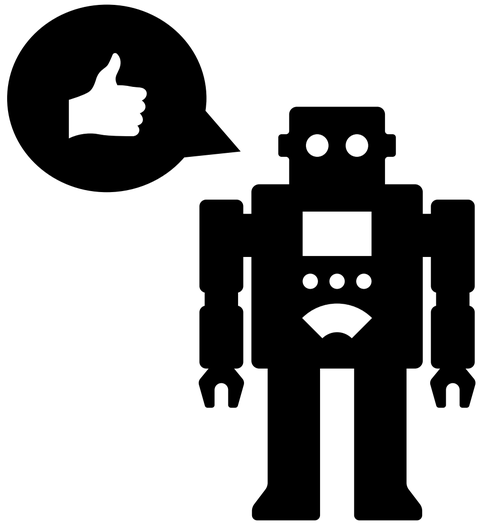
noise

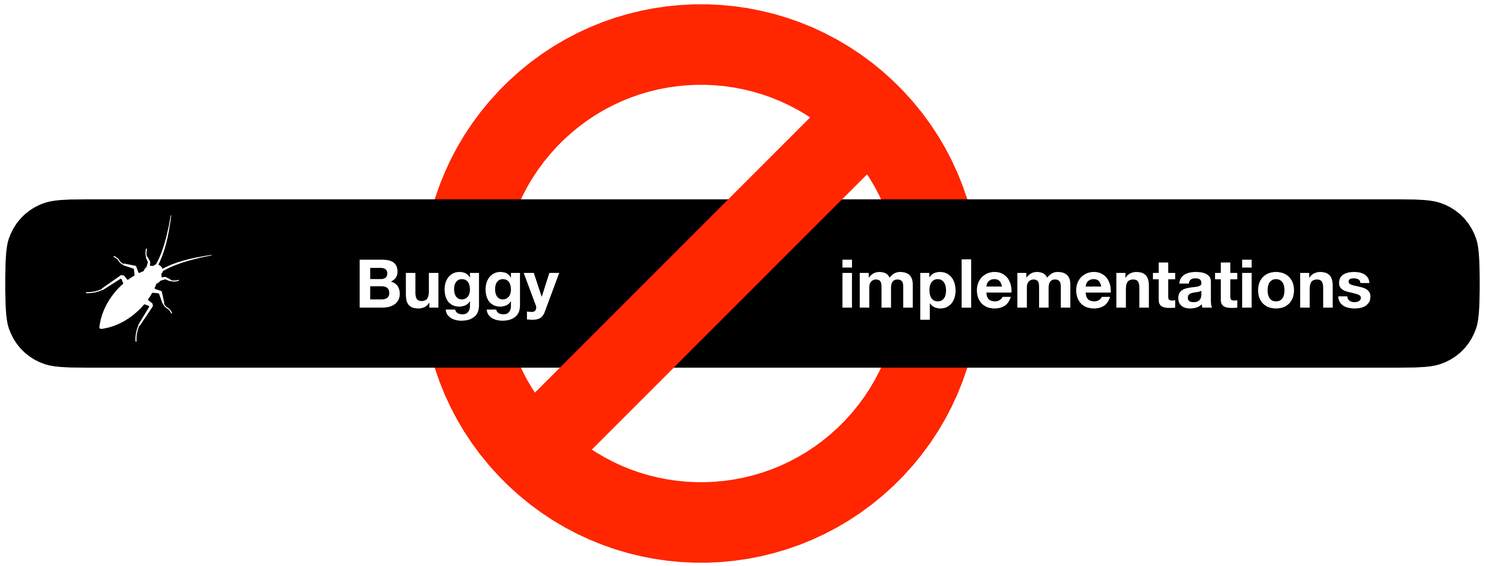
postprocessing



Advanced Privacy Proofs

- zCDP → pure DP conversion bound
- zCDP → approximate DP conversion bound
- pure DP of the sparse vector technique

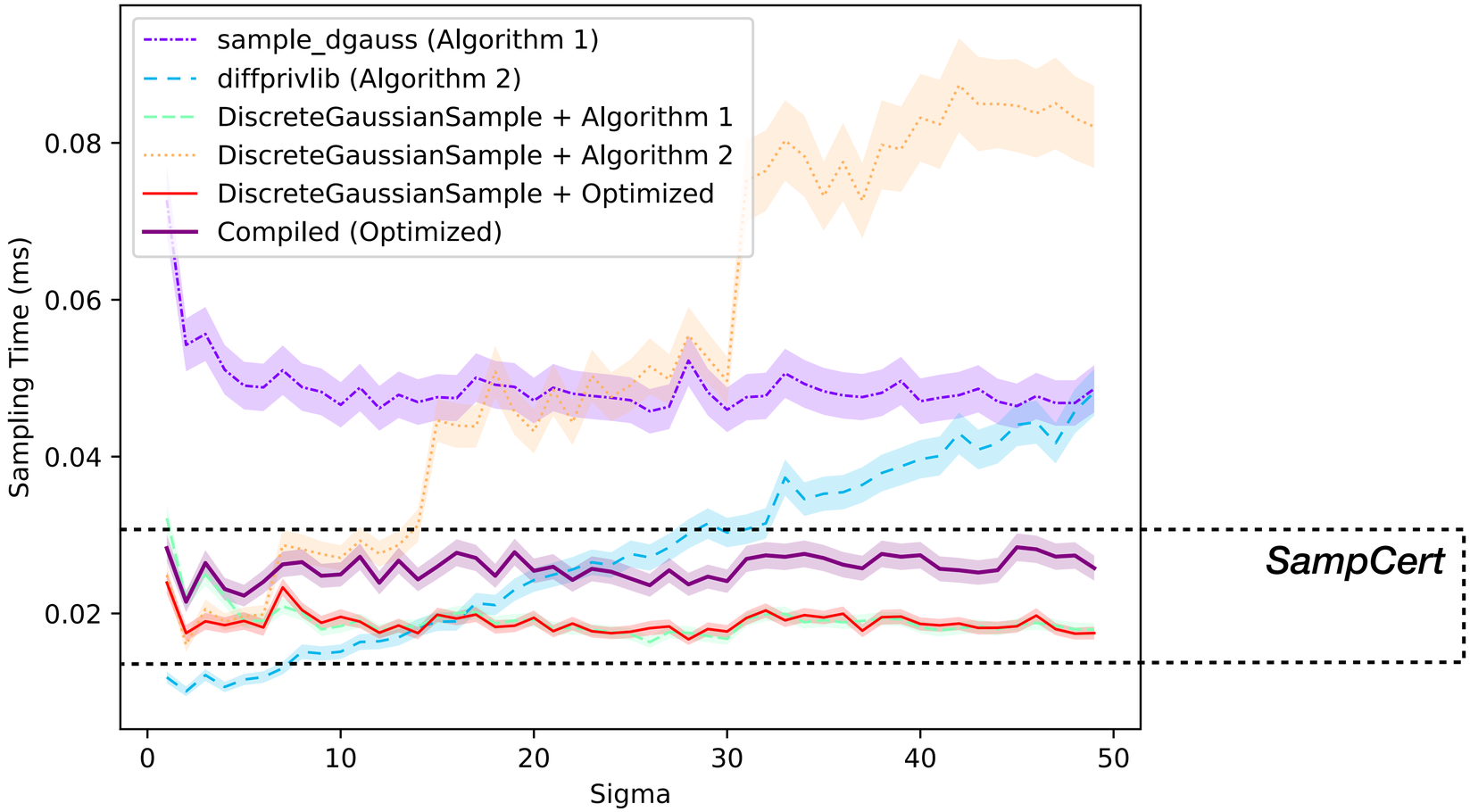




Non-private SVT implementations fail here

Executing SampCert code

Realistic performance



Lean FFI

Return

```
return v;
```

Uniform

```
unsigned char r;
read (random, &r, 1);
return r;
```

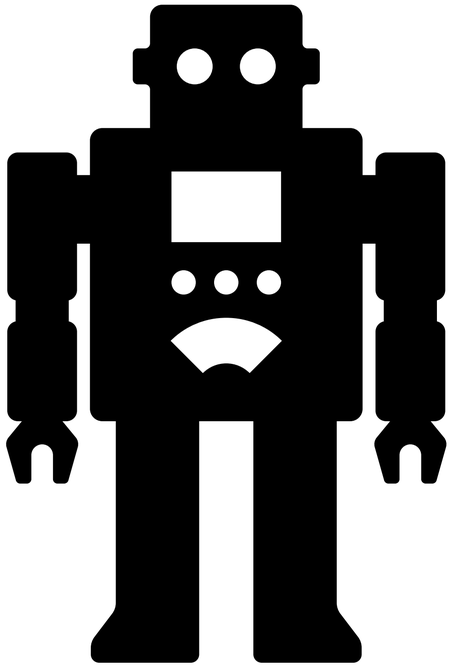
While

```
lean_object* st = init;
uint8_t c = lean_apply_1 (c, st);
while (c) {
    s = lean_apply_2 (b, st);
    c = lean_apply_1 (c, st); }
return st;
```

Bind

```
lean_object* e = lean_apply_1 f;
lean_object* p =
  lean_apply_2 (p, e);
return p;
```

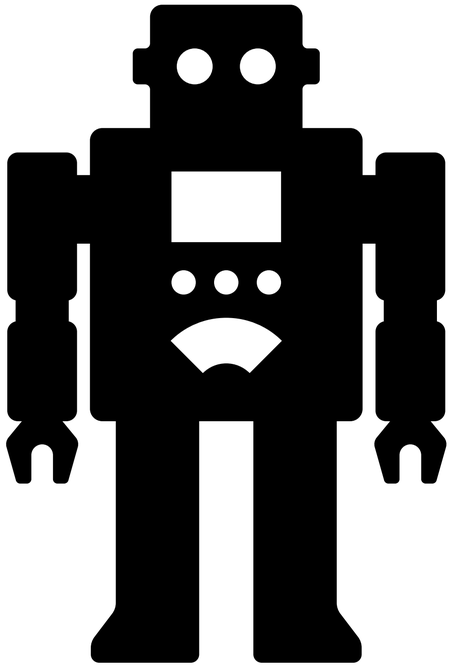
Execution is not verified



✓ High-assurance proof checker

? Conventional compiler, but not formally verified

Execution is not verified



✓ High-assurance proof checker

? Conventional compiler, but not formally verified

Ongoing work: **Incomplete attacker models**

SampCert

Foundations for Verified Differential Privacy

Paper

Verified Foundations for Differential Privacy
(PLDI 2025)

Repo

`github.com/leanprover/SampCert`

PR's Welcome!