MLightDP

Steffan Christ Sølvsten and Anna Mie Hansen

2nd of January, 2020

AARHUS
UNIVERSITY

Definition (Differential Privacy)
Given parameters $\epsilon \geq 0$ then a probabilistic algorithm $M : A \mapsto B$ is ϵ-differentially private if

$$\begin{aligned} &\forall a_1, a_2 \in A \; \forall O \subset B : \\ &\qquad a_1 \Psi a_2 \implies \Pr\left[M(a_1) \in O\right] \leq e^{\epsilon} \Pr\left[M(a_2) \in O\right] \quad . \end{aligned}$$

Randomness Alignment

Consider $q^{(1)}, q^{(2)} \in \mathbb{R}$, where

$$-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1 \ \ .$$

Randomness Alignment

Consider $q^{(1)}, q^{(2)} \in \mathbb{R}$, where

$$-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1 \ .$$

If $\eta_1 \sim Lap(1/\epsilon)$ for $q^{(1)}$ we'd like a $\eta_2 \sim Lap(1/\epsilon)$ for $q^{(2)}$ such that

$$\eta_2 = f(\eta_1) = \eta_1 + \hat{q} \ ,$$

because then

$$q^{(1)} + \eta_1 = q^{(1)} + \eta_1 + \hat{q} = q^{(2)} + \eta_2 \ ,$$

Randomness Alignment

Consider $q^{(1)}, q^{(2)} \in \mathbb{R}$, where

$$-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1 \ .$$

If $\eta_1 \sim Lap(1/\epsilon)$ for $q^{(1)}$ we'd like a $\eta_2 \sim Lap(1/\epsilon)$ for $q^{(2)}$ such that

$$\eta_2 = f(\eta_1) = \eta_1 + \hat{q} \ ,$$

because then

$$q^{(1)} + \eta_1 = q^{(1)} + \eta_1 + \hat{q} = q^{(2)} + \eta_2 \ ,$$

and then

$$\Pr\left[q^{(1)} + Lap(1/\epsilon) \in O\right] \leq e^{\epsilon} \Pr\left[q^{(2)} + Lap(1/\epsilon) \in O\right] \ .$$

LightDP

Based on the above, Zhang and Kifer '17 propose a type system *LightDP* with dependent types, to prove differential privacy of algorithms.

$$\tau ::= \textit{num}_{\mathbb{d}} \mid \textit{num}_{*} \mid \textit{bool}_0 \mid \textit{list}\ \tau \mid \tau_1 \to \tau_2$$

The number type also contains the numerical *distances* between the two executions.

LightDP

A program is differentially private, if

LightDP

A program is differentially private, if

- The returned value has distance 0

$$\frac{\Gamma \vdash e : \mathcal{B}_0 \text{ or } \Gamma \vdash e : \mathit{list}\ \mathcal{B}_0 \text{ or } \ldots}{\Gamma \vdash \mathit{return}\ e \rightharpoonup \mathit{return}\ e} \text{ (T-Return)}^{1}$$

[1] These rules are variations of the ones by Zhang and Kifer, to remove the disconnect between the rule and their/our implementation.

LightDP

A program is differentially private, if

- The returned value has distance 0

$$\frac{\Gamma \vdash e : \mathcal{B}_0 \text{ or } \Gamma \vdash e : \textit{list } \mathcal{B}_0 \text{ or } \ldots}{\Gamma \vdash \textit{return } e \rightharpoonup \textit{return } e} \text{ (T-Return)}^1$$

- The privacy budget variable v_ϵ is less or equal to ϵ in the *transformed* program on all return paths.

$$\frac{\Gamma(\eta) = \textit{num}_{\mathbb{d}} \qquad e : \textit{num}_0}{\Gamma \vdash \eta := \; \textit{Lap}(e) \rightharpoonup \textit{havoc } \eta;\; v_\epsilon := v_\epsilon + |\mathbb{d}|/e} \text{ (T-Laplace)}^1$$

[1] These rules are variations of the ones by Zhang and Kifer, to remove the disconnect between the rule and their/our implementation.

MLightDP

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

```

LaplaceMechanism(q: float)
```

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

```

      LaplaceMechanism(q: float)
      {
        let eta : float       = Lap(1 / epsilon);
```

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

```

LaplaceMechanism(q: float)
{
  let eta : float      = Lap(1 / epsilon);
  return q + eta;
}
```

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

```
precondition: -1.0 <= ^q /\ ^q <= 1.0
LaplaceMechanism(q: float[^q], ^q : float[0])
{
  let eta : float      = Lap(1 / epsilon);
  return q + eta;
}
```

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \le q^{(1)} - q^{(2)} = \hat{q} \le 1$ and adding $\eta \sim \mathit{Lap}(1/\epsilon)$ to q makes them differentially private.

```
precondition: -1.0 <= ^q /\ ^q <= 1.0
LaplaceMechanism(q: float[^q], ^q : float[0])
{
  let eta : float[-^q] = Lap(1 / epsilon);
  return q + eta;
}
```

Example: `laplace_mechanism.mldp`

Remember, the randomness alignment example from before where $-1 \leq q^{(1)} - q^{(2)} = \hat{q} \leq 1$ and adding $\eta \sim Lap(1/\epsilon)$ to q makes them differentially private.

```
precondition: -1.0 <= ^q /\ ^q <= 1.0
LaplaceMechanism(q: float[*])
{
  let eta : float[-^q] = Lap(1 / epsilon);
  return q + eta;
}
```

Example: `laplace_mechanism.dfy`

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;
```

Example: `laplace_mechanism.dfy`

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;

  assert 0.0 == 0.0; // T-Laplace
  assert 0.0 == 0.0; // T-OTimes
  assert 0.0 == 0.0; // T-OTimes
```

Example: `laplace_mechanism.dfy`

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;

  assert 0.0 == 0.0; // T-Laplace
  assert 0.0 == 0.0; // T-OTimes
  assert 0.0 == 0.0; // T-OTimes
  var eta :| 0.0 <= eta;
  if * { eta := eta; } else { eta := -eta; }
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;

  assert 0.0 == 0.0; // T-Laplace
  assert 0.0 == 0.0; // T-OTimes
  assert 0.0 == 0.0; // T-OTimes
  var eta :| 0.0 <= eta;
  if * { eta := eta; } else { eta := -eta; }
  v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);
```

Example: `laplace_mechanism.dfy`

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;

  assert 0.0 == 0.0; // T-Laplace
  assert 0.0 == 0.0; // T-OTimes
  assert 0.0 == 0.0; // T-OTimes
  var eta :| 0.0 <= eta;
  if * { eta := eta; } else { eta := -eta; }
  v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);

  assert q_hat0 + -q_hat0 == 0.0; // T-Return
  assert v_epsilon <= epsilon; // T-Return
```

Example: `laplace_mechanism.dfy`

The transformed *Dafny* program then is:

```
method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
  returns (_:real)
  requires epsilon > 0.0
  requires -1.0 <= q_hat0 && q_hat0 <= 1.0
{
  ghost var v_epsilon : real := 0.0;

  assert 0.0 == 0.0; // T-Laplace
  assert 0.0 == 0.0; // T-OTimes
  assert 0.0 == 0.0; // T-OTimes
  var eta :| 0.0 <= eta;
  if * { eta := eta; } else { eta := -eta; }
  v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);

  assert q_hat0 + -q_hat0 == 0.0; // T-Return
  assert v_epsilon <= epsilon; // T-Return
  return q + eta;
}
```

Examples

Verified?	Example Program
✓	`laplace_mechanism.mldp`
✓	`sparse_vector.mldp` of Zhang and Kifer ’17
✓[1]	`numerical_sparse_vector.mldp` of Zhang and Kifer ’17
✓[1]	`gap_sparse_vector.mldp` of Wang et. al. ’19
	`partial_sum.mldp` of Zhang and Kifer ’17
	`smart_sum.mldp` of Zhang and Kifer ’17

[1]We had to modify the given invariants to make these work.

Comparison

LightDP	*MLightDP*	Feature
✓	✓	DP Type checking
	✓	Output to a Software Verification tool
	✓	Semantic Analysis of distances
		Refinement: Distance inference
	–	Refinement: Mutable Dependency tracking
	–	Free usage of the Laplace
		Automating non-linearity
	–	Other

MLightDP

The Goal of our project is to create a less prototypical experience than their current prototype.

1. Create more intuitive typing annotations.
2. Be capable of accommodating the complete pipeline.
3. Dispense with assumptions of Zhang and Kifer about the input.
4. Make their rules of more fine-grained / less conservative.
5. Soundly add new operators, functions, and alternative random distributions to the language.