

MLightDP

Steffan Christ Sølvesten and Anna Mie Hansen

2nd of January, 2020



Definition (Differential Privacy)

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

$$\forall a_1, a_2 \in A \forall O \subset B :$$

$$a_1 \Psi a_2 \implies \Pr [M(a_1) \in O] \leq e^\epsilon \Pr [M(a_2) \in O] \ .$$

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

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

LightDP

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

$$\tau ::= num_{\mathbb{d}} \mid num_* \mid bool_0 \mid list\ \tau \mid \tau_1 \rightarrow \tau_2$$

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

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

¹These 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

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

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

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

¹These 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 $\eta \sim \text{Lap}(1/\epsilon)$ 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 $\eta \sim \text{Lap}(1/\epsilon)$ to q makes them differentially private.

```
1  
2      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 $\eta \sim \text{Lap}(1/\epsilon)$ to q makes them differentially private.

```
1
2   LaplaceMechanism(q: float)
3   {
4       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 $\eta \sim \text{Lap}(1/\epsilon)$ to q makes them differentially private.

```
1
2   LaplaceMechanism(q: float)
3   {
4       let eta : float      = Lap(1 / epsilon);
5       return q + eta;
6   }
```


Example: laplace_mechanism.mldp

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

```
1      precondition: -1.0 <= ^q /\ ^q <= 1.0
2      LaplaceMechanism(q: float[^q], ^q : float[0])
3      {
4          let eta : float          = Lap(1 / epsilon);
5          return q + eta;
6      }
```

Example: `laplace_mechanism.mldp`

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

```
1      precondition: -1.0 <= ^q /\ ^q <= 1.0
2      LaplaceMechanism(q: float[^q], ^q : float[0])
3      {
4          let eta : float[-^q] = Lap(1 / epsilon);
5          return q + eta;
6      }
```

Example: `laplace_mechanism.mldp`

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

```
1      precondition: -1.0 <= ^q /\ ^q <= 1.0
2      LaplaceMechanism(q: float[*])
3      {
4          let eta : float[-^q] = Lap(1 / epsilon);
5          return q + eta;
6      }
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1    method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2      returns (_, real)
3      requires epsilon > 0.0
4      requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5    {
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1    method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2      returns (_, real)
3      requires epsilon > 0.0
4      requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5    {
6      ghost var v_epsilon : real := 0.0;
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1    method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2      returns (_, real)
3      requires epsilon > 0.0
4      requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5    {
6      ghost var v_epsilon : real := 0.0;
7
8      assert 0.0 == 0.0; // T-Laplace
9      assert 0.0 == 0.0; // T-OTimes
10     assert 0.0 == 0.0; // T-OTimes
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1  method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2      returns (_, real)
3      requires epsilon > 0.0
4      requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5  {
6      ghost var v_epsilon : real := 0.0;
7
8      assert 0.0 == 0.0; // T-Laplace
9      assert 0.0 == 0.0; // T-OTimes
10     assert 0.0 == 0.0; // T-OTimes
11     var eta :| 0.0 <= eta;
12     if * { eta := eta; } else { eta := -eta; }
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1  method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2    returns (_, real)
3    requires epsilon > 0.0
4    requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5  {
6    ghost var v_epsilon : real := 0.0;
7
8    assert 0.0 == 0.0; // T-Laplace
9    assert 0.0 == 0.0; // T-OTimes
10   assert 0.0 == 0.0; // T-OTimes
11   var eta :| 0.0 <= eta;
12   if * { eta := eta; } else { eta := -eta; }
13   v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);
```


Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1  method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2    returns (_, real)
3    requires epsilon > 0.0
4    requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5  {
6    ghost var v_epsilon : real := 0.0;
7
8    assert 0.0 == 0.0; // T-Laplace
9    assert 0.0 == 0.0; // T-OTimes
10   assert 0.0 == 0.0; // T-OTimes
11   var eta :| 0.0 <= eta;
12   if * { eta := eta; } else { eta := -eta; }
13   v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);
14
15   assert q_hat0 + -q_hat0 == 0.0; // T-Return
16   assert v_epsilon <= epsilon; // T-Return
```

Example: laplace_mechanism.dfy

The transformed *Dafny* program then is:

```
1  method LaplaceMechanism(epsilon : real, q : real, ghost q_hat0 : real)
2    returns (_, real)
3    requires epsilon > 0.0
4    requires -1.0 <= q_hat0 && q_hat0 <= 1.0
5  {
6    ghost var v_epsilon : real := 0.0;
7
8    assert 0.0 == 0.0; // T-Laplace
9    assert 0.0 == 0.0; // T-OTimes
10   assert 0.0 == 0.0; // T-OTimes
11   var eta :| 0.0 <= eta;
12   if * { eta := eta; } else { eta := -eta; }
13   v_epsilon := v_epsilon + Abs(-q_hat0) / ((1 as real) / epsilon);
14
15   assert q_hat0 + -q_hat0 == 0.0; // T-Return
16   assert v_epsilon <= epsilon; // T-Return
17   return q + eta;
18 }
```

Examples

| Verified? | Example Program |
|----------------|--|
| ✓ | <code>laplace_mechanism.mldp</code> |
| ✓ | <code>sparse_vector.mldp</code> of Zhang and Kifer '17 |
| ✓ ¹ | <code>numerical_sparse_vector.mldp</code> of Zhang and Kifer '17 |
| ✓ ¹ | <code>gap_sparse_vector.mldp</code> of Wang et. al. '19 |
| | <code>partial_sum.mldp</code> of Zhang and Kifer '17 |
| | <code>smart_sum.mldp</code> of Zhang and Kifer '17 |

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

Comparison

| <i>LightDP</i> | <i>MLightDP</i> | 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.