LightDP: Towards Automating Differential Privacy Proofs

Danfeng Zhang    Daniel Kifer
Penn State University
Alice’s data remain private if $\mu_1, \mu_2$ are close
(Pure) Differential Privacy

If for any adjacent databases and value \( \nu \), \( \mu_1(\nu)/\mu_2(\nu) \leq e^\epsilon \) for some constant \( \epsilon \), then a computation is \( \epsilon \)-private.
Motivation

DP has seen explosive growth since 2006
- U.S. Census Bureau LEHD OnTheMap tool [Machanavajjhala et al. 2008]
- Google Chrome Browser [Erlingsson et al. 2014]
- Apple’s new data collection efforts [Greenberg 2016]

But also accompanied with flawed (paper-and-pencil) proofs
- e.g., ones categorized in [Chen&Machanavajjhala’15, Lyu et al.’16]

Rigorous methods are needed for differential privacy proofs
Related Work

DP programming platforms (e.g., PINQ, Airavat)
  • Use (instead of verify) basic DP mechanisms
  • Cannot offer tight bounds for sophisticated algorithms

Methods based on customized logics
  • Steep learning curve
  • Heavy annotation burden

LightDP offers a better balance between expressiveness and usability
LightDP: Overview

Source Program

Source program type checks

Target Program with distinguished variable $V_\varepsilon$

$v_\varepsilon := 0$; 
$havoc \ \eta_1; v_\varepsilon := v_\varepsilon + \varepsilon/2; 
\tilde{T} := T + \eta_1; 
c1 := 0; c2 := 0; i := 0; 
while (c1 < N) 
\eta_2 := \text{Lap}(4N/\varepsilon); 
if (q[i] + \eta_2 \geq \tilde{T}) then 
\text{out} := \text{true}; \text{out}; 
c1 := c1 + 1; 
else 
\text{out} := \text{false}; \text{out}; 
c2 := c2 + 1; 
i := i+1;

Relational, Dependent Type System

Main Theorem

Source program is $\varepsilon$-private

$v_\varepsilon$ bounded by constant $\varepsilon$ in the target program
Source Language: Syntax

Commands: $c ::= \text{skip} \mid x := e \mid \eta := g \mid c_1 ; c_2 \mid \text{return } e \mid \text{if } e \text{ then } c_1 \text{ else } c_2 \mid \text{while } e \text{ do } c$
Source Language: Semantics

Memory: mapping from variables to values

Initial memory

Adjacent memory

Relational Reasoning via Type System

Final memory dist.
Relational Types

\[ B_d \]

Example
\[ \Gamma(x): \text{num}_0 \]
\[ \Gamma(y): \text{num}_1 \]

Base Type: e.g., int, real

Related Memories
\[ x: u \]
\[ y: v \]

Distance
\[ x: u \]
\[ y: v+1 \]
Dependent Types

\[ B_d \]

Can be a program variable

Example

\[ \Gamma(x): \text{num}_0 \]
\[ \Gamma(y): \text{num}_x \]

Related Memories

\[ x: u \]
\[ y: v \]

\[ x: u \]
\[ y: v + u \]
Dependent Types

Can be a non-prob. expression

Example

\[ \Gamma(x) : \text{num}_0 \]
\[ \Gamma(y) : \text{num}_{x \geq 1 ? 2 : 0} \]

Related Memories

\[ x : u \]
\[ y : \begin{cases} v + 2, u \geq 1 \\ v, u < 1 \end{cases} \]

Notation

\[ m_1 \Gamma m_2 \text{ if } m_1 \text{ and } m_2 \text{ are related by } \Gamma \]
(for the non-probabilistic subset)
Types form an invariant on two related program executions:

If initial memories $m_1$

Then after executing a well-typed program, final memories $m'_1$

Enforced by a type system
Type System

Expression: \( \Gamma \vdash e : \mathcal{B}_d \)

\[
\Gamma \vdash e_1 : \text{num}_{d_1} \quad \Gamma \vdash e_2 : \text{num}_{d_2} \\
\Gamma \vdash e_1 \oplus e_2 : \text{num}_{d_1 \oplus d_2}
\]

\[
\Gamma \vdash e_1 : \text{num}_{d_1} \\
\Gamma \vdash e_2 : \text{num}_{d_2} \\
\Gamma \vdash (e_1 + d_1) \otimes (e_2 + d_2) \iff (e_1 \ominus e_2) \\
\Gamma \vdash e_1 \ominus e_2 : \text{bool}_0
\]
Type System

Command:

\[ \Gamma \vdash c \]

\[ \Gamma \vdash e : \tau \quad \Gamma \vdash x : B_d \quad \tau = B_d \]

\[ \Gamma \vdash x := e \]

Distance must be identical

Related executions take same branch

\[ \Gamma \vdash e : \text{bool}_0 \quad \Gamma \vdash c_i \text{ where } i \in \{1, 2\} \]

\[ \Gamma \vdash \text{if } e \text{ then } c_1 \text{ else } c_2 \]
Relating Two Distributions

$$\mu_1 \Gamma \mu_2 \text{ w.r.t. privacy cost } \epsilon \text{ if }$$

$$\forall m. \mu_1(m)/\mu_2(\Gamma(m)) \leq e^\epsilon$$

Program

$$\eta := \text{Lap } r$$

$$\Gamma(\eta) = \text{num}_0$$

Laplace dist. w/ mean 0 and a scale factor $$r$$

With no cost
Relating Two Distributions

\( \mu_1 \triangleright \mu_2 \) w.r.t. privacy cost \( \epsilon \) if
\[
\forall m. \frac{\mu_1(m)}{\mu_2(\Gamma(m))} \leq e^\epsilon
\]

Program
\[
\eta := \text{Lap} \ r
\]
\[
\Gamma(\eta) = \text{num}_1
\]

Observation
\( \eta \) may have an arbitrary distance, which affects the added cost

With cost \( 1/r \) due to dist. property
\[ \Gamma(\eta) = \text{num}_d \]

\[ \Gamma \vdash \eta := \text{Lap} \ r \]

\( \eta \) has a polymorphic type

Non-deterministic operation

\( \eta \) may have an arbitrary distance, which affects the added cost

Intuitively, target program computes the added cost for one sample from distribution

Source program

Target program, explicitly tracks added privacy cost
In General

Source program

\[
\eta_1 := \text{Lap}(2/\epsilon); \\
\tilde{T} := T + \eta_1; \\
c1 := 0; \ c2 := 0; \ i := 0; \\
\text{while} \ (c1 < N) \\
\eta_2 := \text{Lap}(4N/\epsilon); \\
\text{if} \ (q[i] + \eta_2 \geq \tilde{T}) \ \text{then} \\
\quad \text{out} := \text{true}; \out; \\
\quad c1 := c1 + 1; \\
\text{else} \\
\quad \text{out} := \text{false}; \out; \\
\quad c2 := c2 + 1; \\
\quad i := i + 1;
\]

Target program with distinguished variable \(V_\epsilon\)

\[
v_\epsilon := 0; \\
havoc \ \eta_1; v_\epsilon := v_\epsilon + \epsilon/2; \\
\tilde{T} := T + \eta_1; \\
c1 := 0; \ c2 := 0; \ i := 0; \\
\text{while} \ (c1 < N) \\
\text{havoc} \ \eta_2; v_\epsilon := v_\epsilon + (q[i] + \eta_2 \geq \tilde{T})?0 \times \epsilon/4N; \\
\text{if} \ (q[i] + \eta_2 \geq \tilde{T}) \ \text{then} \\
\quad \text{out} := \text{true}; \out; \\
\quad c1 := c1 + 1; \\
\text{else} \\
\quad \text{out} := \text{false}; \out; \\
\quad c2 := c2 + 1; \\
\quad i := i + 1;
\]

Type System

\[\Gamma \vdash c \rightarrow c'\]

source program

target program
Target Language

Verification task in the target language:
Proving $V_\epsilon$ is bounded by some constant $\epsilon$ in any execution (in a non-probablistic program)

A safety property. Can be verified using off-the-shelf tools (e.g., Hoare logic, model checking)
Putting Together
The Sparse Vector Method [Dwork and Roth’14]

Source Program

\[\eta_1 := \text{Lap}\left(\frac{2}{\epsilon}\right);\]
\[\tilde{T} := T + \eta_1;\]
\[c_1 := 0; \quad c_2 := 0; \quad i := 0;\]
\[\text{while } (c_1 < N)\]
\[\quad \eta_2 := \text{Lap}\left(\frac{4N}{\epsilon}\right);\]
\[\quad \text{if } (q[i] + \eta_2 \geq \tilde{T})\]
\[\quad \quad \text{out} := \text{true} ; \quad c_1 := c_1 + 1;\]
\[\quad \text{else}\]
\[\quad \quad \text{out} := \text{false}; \quad c_2 := c_2 + 1;\]
\[i := i + 1;\]

• Correctness proof is subtle
  Incorrect variants categorized in
  [Chen&Machanavajjhala’15, Lyu et al.’16]

• Formally verified very
  recently [Barthe et al. 2016]
  with heavy annotation burden
Required Types

\[ c_1, c_2, i : \text{num}_0; \tilde{T}, \eta_1 : \text{num}_1; \eta_2 : \text{num} \quad q[i] + \eta_2 \geq \tilde{T} ? 2:0 \]

\[ \eta_1 := \text{Lap} \left( \frac{2}{\epsilon} \right); \]
\[ \tilde{T} := T + \eta_1; \]
\[ c_1 := 0; \quad c_2 := 0; \quad i := 0; \]
\[ \textbf{while} \quad (c_1 < N) \quad \]
\[ \quad \eta_2 := \text{Lap} \left( \frac{4N}{\epsilon} \right); \]
\[ \quad \textbf{if} \quad (q[i] + \eta_2 \geq \tilde{T}) \quad \textbf{then} \]
\[ \quad \quad \text{out} := \text{true}; \quad \text{out}; \]
\[ \quad \quad c_1 := c_1 + 1; \]
\[ \quad \textbf{else} \]
\[ \quad \quad \text{out} := \text{false}; \quad \text{out}; \]
\[ \quad \quad c_2 := c_2 + 1; \]
\[ \quad i := i + 1; \]

Distance depends on the value of \( i \)th query answer \( (q[i]) \)

**Type Inference**

Types can be inferred by the inference algorithm of LightDP
Target Program

\[
\begin{align*}
\eta_1 &:= \text{Lap}(2/\epsilon); \\
\tilde{T} &:= T + \eta_1; \\
c_1 &:= 0; \quad c_2 := 0; \quad i := 0; \\
\text{while } (c_1 < N) \\
\eta_2 &:= \text{Lap}(4N/\epsilon); \\
\text{if } (q[i] + \eta_2 \geq \tilde{T}) \text{ then} \\
\quad \text{out} &:= \text{true}::\text{out}; \\
\quad c_1 &:= c_1 + 1; \\
\text{else} \\
\quad \text{out} &:= \text{false}::\text{out}; \\
\quad c_2 &:= c_2 + 1; \\
i &:= i+1;
\end{align*}
\]
Completing the Proof

\[ \mathbf{v}_\varepsilon := 0; \]
\[ \text{havoc } \eta_1; \mathbf{v}_\varepsilon := \mathbf{v}_\varepsilon + \varepsilon/2; \]
\[ \tilde{T} := T + \eta_1; \]
\[ c_1 := 0; \quad c_2 := 0; \quad i := 0; \]
\[ \textbf{while } (c_1 < N) \]
\[ \quad \textbf{Invariant: } c_1 \leq N \land \mathbf{v}_\varepsilon = \varepsilon/2 + c_1 \times \frac{\varepsilon}{2N} \]
\[ \text{havoc } \eta_2; \mathbf{v}_\varepsilon := \mathbf{v}_\varepsilon + (q[i] + \eta_2 \geq \tilde{T} \Rightarrow 2 : 0) \times \varepsilon/4N; \]
\[ \textbf{if } (q[i] + \eta_2 \geq \tilde{T}) \textbf{ then} \]
\[ \quad \text{out} := \text{true}; \]
\[ \quad c_1 := c_1 + 1; \]
\[ \textbf{else} \]
\[ \quad \text{out} := \text{false}; \]
\[ \quad c_2 := c_2 + 1; \]
\[ \quad i := i + 1; \]

Postcondition: \( \mathbf{v}_\varepsilon \leq \varepsilon \)

Loop Invariant

Main Theorem

Source program type checks + \( \mathbf{V}_\varepsilon \) bounded by constant \( \varepsilon \)
= source program is \( \varepsilon \)-private
More in the Paper

Type inference algorithm

Searching for proof with minimum cost w/ MaxSMT

Formal proof for the main theorem

More verified examples (with little manual efforts)
Summary

Automated by inference engine

A safety property (verified by existing tools)

Target Program with distinguished variable \( V_\epsilon \)

Decomposing differential privacy into subtasks substantially simplifies language-based proof

Source Program

Relational, Dependent Type System

Program

\[
\eta_1 \ := \ \text{Lap} \left( \frac{2}{\epsilon} \right); \\
\bar{T} \ := \ T + \eta_1; \\
c1 \ := \ 0; \ c2 \ := \ 0; \ i \ := \ 0; \\
\text{while } (c1 < N) \ \\
\eta_2 \ := \ \text{Lap} \left( \frac{4N}{\epsilon} \right); \\
\text{if } (q[i] + \eta_2 \geq \bar{T}) \ \\
\text{then } \\
\text{out} := \text{true}; \text{out}; \\
c1 \ := \ c1 + 1; \\
\text{else} \\
\text{out} := \text{false}; \text{out}; \\
c2 \ := \ c2 + 1; \\
i \ := \ i+1;
\]

Program

\[
v_\epsilon \ := \ 0; \\
havoc \ \eta_1; v_\epsilon := v_\epsilon + \epsilon/2; \\
\bar{T} := T + \eta_1; \\
c1 := 0; c2 := 0; i := 0; \\
\text{while } (c1 < N) \ \\
havoc \ \eta_2; v_\epsilon := v_\epsilon + (q[i] + \eta_2 \geq \bar{T} ? 0 : \epsilon/4N); \\
\text{if } (q[i] + \eta_2 \geq \bar{T}) \ \\
\text{then } \\
\text{out} := \text{true}; \text{out}; \\
c1 := c1+1; \\
\text{else} \\
\text{out} := \text{false}; \text{out}; \\
c2 := c2+1; \\
i := i+1;
\]