Automatic Language-Based Verification of Differential Privacy

by

Chiké Abuah

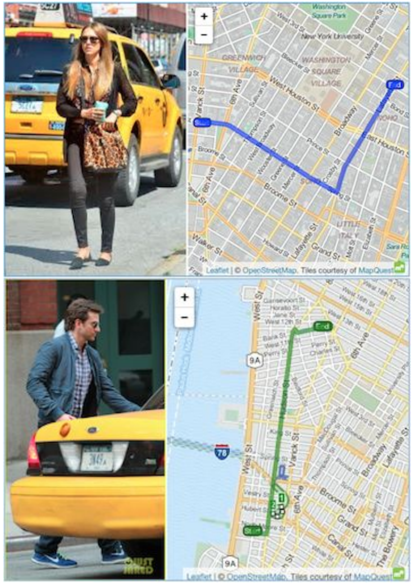
Goal: reveal trends in the population, without revealing information about individuals

Why Do We Need Differential Privacy?

NETFLIX + IMDb

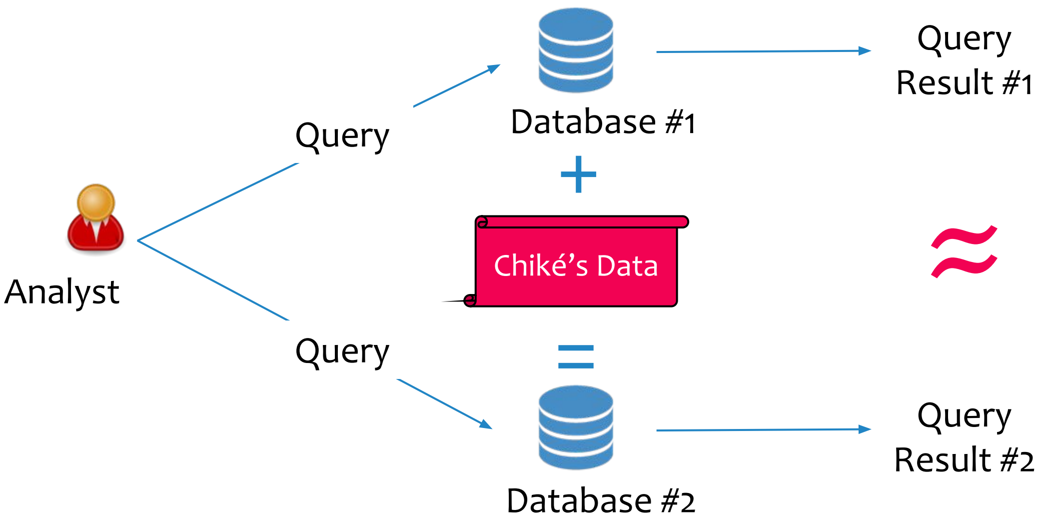
- Ad-hoc privacy techniques (e.g. anonymization) can't guarantee privacy

- Subject to re-identification attacks
    - Netflix recommender algorithm prize (Narayanan et al.)
    - NYC taxi data (Anthony Tockar)



Differential Privacy is a Definition of Privacy

- Adversary can’t tell from analysis result whether or not Chiké participated
- Guarantee parameterized by ε (the privacy budget)



Satisfying Differential Privacy

To satisfy differential privacy, **add noise to the query result**

$$F(x) = f(x) + \textbf{Noise}$$

How much noise? **It depends on f**

How many trips were taken in New York last year?	If Chiké lives in Vermont, return 100000; else return 0
Low sensitivity	**High sensitivity**

Why do we need Verification?

Algorithm 1 $\mathcal{A}_{\text{Noise-GD}}$: Differentially Private Gradient Descent

Input: Data set: $\mathcal{D} = \{d_1, \cdots, d_n\}$, loss function ℓ (with Lipschitz constant L), privacy parameters (ϵ, δ), convex set $\mathcal{C}$, and the learning rate function $\eta : [n^2] \rightarrow \mathbb{R}$.

Set noise variance $\sigma^2 \leftarrow \boxed{O\left(\frac{L^2 n^2 \log(n/\delta)\log(1/\delta)}{\epsilon^2}\right)}$.

$\widetilde{\theta}_1$: Choose any point from $\mathcal{C}$.

for $t = 1$ to $n^2 - 1$ **do**

  Pick $d \sim_u \mathcal{D}$ with replacement.

  $\widetilde{\theta}_{t+1} = \Pi_{\mathcal{C}}\left(\widetilde{\theta}_t - \eta(t)\left[n \triangledown \ell(\widetilde{\theta}_t; d) + b_t\right]\right)$, $b_t \sim \mathcal{N}\left(0, \mathbb{I}_p \sigma^2\right)$.

Output $\theta^{priv} = \widetilde{\theta}_{n^2}$.

How do we know it's the right amount of noise? *Manual proof*

Incorrect differentially private algorithms ***don't crash***

They ***silently violate your privacy***

Privacy by (programming language) Design!

- Accurate
- Accessible to non-experts
- Automatic
- Ubiquitous
- Context Matters: Support all manner of programs, systems, software, languages, modes of operation, etc.

Program Analysis Techniques

Static Analysis

- Control Flow Analysis
- Data Flow Analysis
- Abstract Interpretation
- Type Systems
- Effect Systems
- Model Checking

Dynamic Analysis

- Testing
- Monitoring (Runtime Verification)
- Program Slicing

Talk Summary: Projects

- **Duet**: PL, linear types for (ε,δ)
- **DDuo**: dynamic analysis system
- **Solo**: static analysis system

Big Picture Practical Significance

- In practice differential privacy can be difficult to analyze. This work proposes several techniques for **automatic analysis** and **convenient implementation** of differential privacy.
- Ideally, little or no domain knowledge should be required from programmers.
- Privacy violations are **silent,** making it an **especially important field in which to apply verification techniques**.

Threat Model

- Assumes an “honest but fallible” programmer.
- Does not address side-channels.
- Terminated programs can be rerun safely (while consuming the privacy budget).

DUET:

An Expressive Language for Statically Verifying Differential Privacy

(Published at OOPSLA 2019)

ACM SIGPLAN Distinguished Paper Award Winner

Duet is ***the first linear typed language to support verification of advanced variants of Differential Privacy***

Duet automatically proves that:

$$
\begin{aligned}
&\mathrm{noisy\text{-}gradient\text{-}descent}(X, y, k, \epsilon, \delta) \triangleq \\
&\quad \texttt{let } \theta_0 = \mathrm{zeros}\ (\mathrm{cols}\ X_1)\ \texttt{in} \\
&\quad \texttt{loop}[\delta\ ]\ k\ \texttt{on}\ \theta_0\ \langle X_1, y\rangle\ \{t, \theta \Rightarrow \\
&\qquad g_p \leftarrow \texttt{mgauss}[\tfrac{1}{m}, \epsilon, \delta]\ \langle X, y\rangle\ \{\mathrm{gradient}\ \theta\ X\ y\}\ ; \\
&\qquad \texttt{return}\ \theta - g_p \quad \}
\end{aligned}
$$

satisfies $(2\epsilon\sqrt{2k\log(1/\delta)}, k\delta + \delta)$ **differential privacy**

Duet's Contributions

- Support for **(ε,δ)-DP and other variants** in a linear type system
- Expressive **matrix types** and API for machine learning
- Ability to **mix privacy variants** within a single program
- Advanced privacy variants are more accurate in most machine learning scenarios

Duet gets its name...

Duet is a co-design of two distinct, mutually embedded type systems:

- one for sensitivity which leverages linear typing *with scaling* a la Fuzz, and
- one for privacy which leverages linear typing ***without scaling*** and is novel in this work

However, to the user, it's all just one language!

Duet’s Two Languages

```
noisy-gradient-descent(X, y, k, ε, δ) ≜
let θ0 = zeros (cols X1) in
loop[δ ] k on θ0 <X1, y> {t, θ ⇒
    gp ← mgauss[1/m, ε, δ] <X, y> {gradient θ X y} ;
    return θ − gp    }
```

Sensitivity Language

Gradient says how to improve model
Sensitivity: 1/m

Add noise to gradient to achieve (ε, δ)-DP

Interface between languages: Gaussian mechanism

Privacy Language

Compose many iterations with advanced composition

Linear Types for Sensitivity

$$x + x$$

2-sensitive in *x*

$$\underbrace{\{x :_2 \mathbb{R}\}}_{\text{Context encodes sensitivity of } x} \vdash x + x : \mathbb{R}$$

$$\frac{}{\{x :_1 \tau\} \vdash x : \tau}$$

$$\frac{\Gamma_1 \vdash e_1 : \mathbb{R} \qquad \Gamma_2 \vdash e_2 : \mathbb{R}}{\underbrace{\Gamma_1 + \Gamma_2}_{\text{Add up sensitivities for each variable}} \vdash e_1 + e_2 : \mathbb{R}}$$

Fuzz: Jason Reed and Benjamin C Pierce. 2010. *Distance makes the types grow stronger: a calculus for differential privacy*. ICFP 45, 9 (2010).

Sensitivity in Functions

$$\emptyset \vdash \underbrace{\lambda x : \mathbb{R} \Rightarrow x + x}_{\text{2-sensitive function in } x} : \underbrace{\mathbb{R} \multimap_2 \mathbb{R}}_{\text{Type of a 2-sensitive function}}$$

Scaling Sensitivity

$$\{y :_4 \mathbb{R}\} \vdash (\lambda x : \mathbb{R} \Rightarrow x + x)\ (y + y) : \mathbb{R}$$

2-sensitive function in *x*

Program is 4-sensitive in *y*

2-sensitive expression in *y*

Application rule *scales* context of the argument by the function's sensitivity

$\multimap$-E

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap_s \tau_2 \qquad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1 + s\Gamma_2 \vdash e_1\ e_2 : \tau_2}$$

Linear Types for Privacy

$$\underbrace{\{x :_{\epsilon,\delta} \mathbb{R}, y :_{\epsilon,\delta} \mathbb{R}\}}_{\text{Context encodes privacy cost for } x \text{ and } y} \vdash \mathrm{gauss}[\mathbb{R}^{+}[1.0], \underbrace{\epsilon, \delta}_{\text{Privacy parameters}}] \underbrace{<x, y>}_{\text{“Important” variables}} \underbrace{\{x + y\}}_{\text{Body expression}} : \mathbb{R}$$

Privacy Scaling is Not Allowed

$$\{y :_{\cancel{2\epsilon,2\delta}} \mathbb{R}\} \vdash (p\lambda(x : \mathbb{R}) \Rightarrow ...)\ (y + y) : \mathbb{R}$$

Scaling is **allowed** in ε-differential privacy ("group privacy")

Used in *Fuzz* (Reed & Pierce 2010) and *DFuzz* (Gaboardi et al. 2013)

Scaling is **not allowed** in (ε,δ)-differential privacy

Duet's Two Languages

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap_s \tau_2 \qquad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1 + s\Gamma_2 \vdash e_1\ e_2 : \tau_2}\ \multimap\text{-E}$$

Sensitivity language:
scaling allowed

Privacy language:
no scaling allowed

$$\frac{\Gamma \vdash e : (\tau_1 @ p_1, \ldots, \tau_n @ p_n) \multimap^* \tau \qquad \rceil\Gamma_1\lceil^1 \vdash e_1 : \tau_1 \quad \cdots \quad \rceil\Gamma_n\lceil^1 \vdash e_n : \tau_n}{\rceil\Gamma\lceil^\infty + \rceil\Gamma_1\lceil^{p_1} + \cdots + \rceil\Gamma_n\lceil^{p_n} \vdash e(e_1, \ldots, e_n) : \tau}\ \multimap^*\text{-E}$$

Theorem (Fundamental Property/Metric Preservation: Sensitivity)

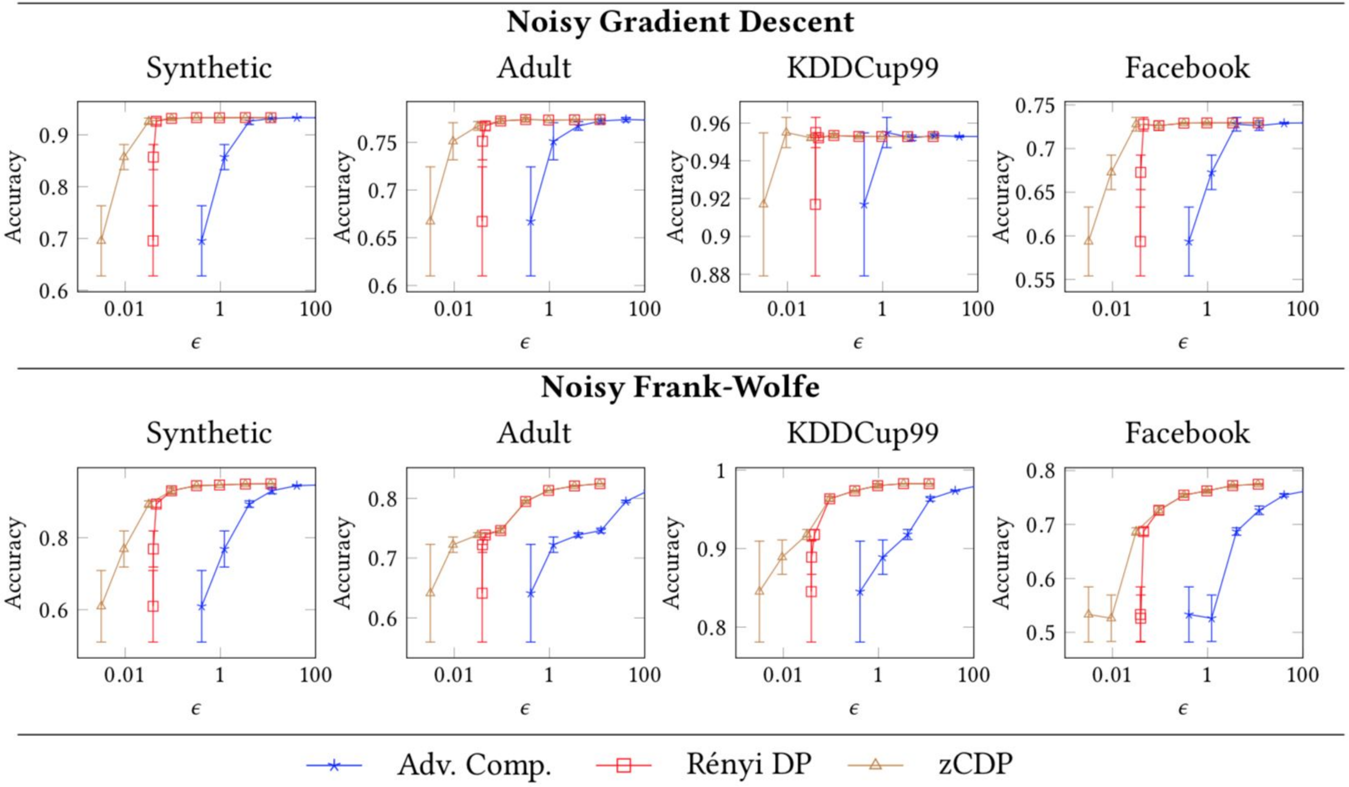
$$\Gamma \vdash e : \tau \; ; \Sigma \quad , \quad (\gamma_1, \gamma_2) \in \mathcal{G}_{\Sigma'}[\![\Gamma]\!] \Rightarrow (\gamma_1 \vdash e, \gamma_2 \vdash e) \in \mathcal{E}_{\Sigma' \cdot \Sigma}[\![\Sigma'(\tau)]\!]$$

$$\boxed{\begin{array}{c} (v, v) \in \mathcal{V}_s[\![\tau]\!] \\ (e, e) \in \mathcal{E}_s[\![\tau]\!] \\ (\gamma, \gamma) \in \mathcal{G}_\Sigma[\![\Gamma]\!] \end{array}}$$

$$\begin{array}{rcl} (r_1, r_2) \in \mathcal{V}_s[\![\mathbb{R}]\!] & \overset{\Delta}{\iff} & |r_1 - r_2| \leq s \\ (\gamma_1 \vdash e_1, \gamma_2 \vdash e_2) \in \mathcal{E}_s[\![\tau]\!] & \overset{\Delta}{\iff} & \forall v_1, v_2, \\ & & \gamma_1 \vdash e_1 \Downarrow v_1 \wedge \gamma_2 \vdash e_2 \Downarrow v_2 \\ & & \implies (v_1, v_2) \in \mathcal{V}_s[\![\tau]\!] \\ (\gamma_1, \gamma_2) \in \mathcal{G}_\Sigma[\![\Gamma]\!] & \overset{\Delta}{\iff} & \forall x \in dom(\Gamma).\ (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}_{\Sigma(x)}[\![\Gamma(x)]\!] \end{array}$$

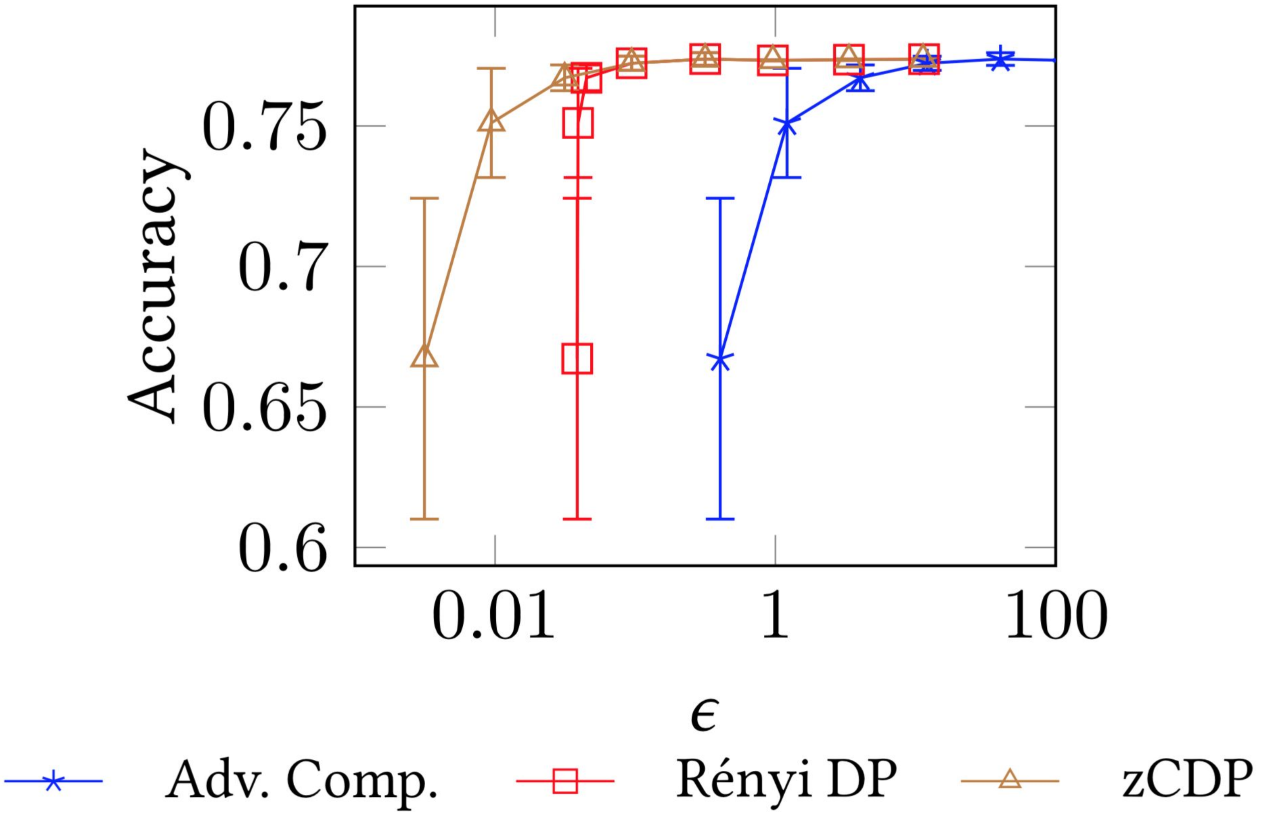
when given related initial configurations and evaluation outputs, then those outputs are related

Empirical Results (Accuracy of Trained Models)



Duet programs produce state-of-the-art accuracy in private linear models

Benefit of Differential Privacy Variants



Recent privacy variants yield **better accuracy** at the **same level of privacy**

Empirical Results (Typechecker Performance)

Technique	LOC	Time (ms)
Noisy G.D.	23	0.51ms
G.D. + Output Pert.	25	0.39ms
Noisy Frank-Wolfe	31	0.59ms
Minibatching	26	0.51ms
Parallel minibatching	42	0.65ms
Gradient clipping	21	0.40ms
Hyperparameter tuning	125	3.87ms
Adaptive clipping	68	1.01ms
Z-Score normalization	104	1.51ms

Typechecker computes privacy cost of complex programs in **milliseconds**

DDUO:

General-Purpose Dynamic Analysis for Differential Privacy

(Published at CSF 2021)

Contributions!

- includes all base types
- general language operations
- various notions of sensitivity
- advanced privacy variants
- generalizes to all future possible arbitrary inputs

DDUO gets its name from...

Product: DDUO makes it easy to automatically enforce privacy

```
def dp_gradient_descent(iterations, alpha, eps):
    eps_i = eps/iterations
    theta = np.zeros(X_train.shape[1])
    noisy_count = duet.renyi_gauss(X_train.shape[0], α = alpha, ε = eps)
    for i in range(iterations):
        grad_sum        = gradient_sum(theta, X_train, y_train, sensitivity)
        noisy_grad_sum  = gaussian_mech_vec(grad_sum, alpha, eps_i)
        noisy_avg_grad  = noisy_grad_sum / noisy_count
        theta           = np.subtract(theta, noisy_avg_grad)
    return theta
```

- Usable
- For non-experts
- Capable of complex algorithms

DDUO is prototyped in Python

- Does not require the programmer to write any additional type annotations.
- In many cases, DDUO can verify differential privacy for essentially unmodified Python programs.
- easily integrated with popular libraries like Pandas and NumPy.

```
from dduo import pandas as pd
df = pd.read_csv("data.csv")
df

# no change to sensitivity environment
df + 5

# doubles the sensitivity
df + df

( df * 5, df * df)
```

```
with dduo.EpsOdometer() as odo:
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  print(odo)
```

DDUO Overview

- DDUO **data sources** are wrappers around sensitive data.
- DDUO tracks the sensitivity of a value to changes in the program's inputs using a sensitivity environment mapping input *data sources* to *sensitivities*.
- Sensitivity environments of `Sensitive` objects update as operations are applied to them (f (x) = x + x).
- DDUO tracks total privacy cost using objects called **privacy odometers**.
- DDUO also allows the analyst to place upper bounds on total privacy cost (i.e. a privacy budget) using **privacy filters**.

```
from dduo import pandas as pd
df = pd.read_csv("data.csv")
df

# no change to sensitivity environment
df + 5

# doubles the sensitivity
df + df

( df * 5, df * df)
```

```
with dduo.EpsOdometer() as odo:
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  print(odo)
```

Sensitivity Analysis: Object Proxies

```python
from dduo import pandas as pd
df = pd.read_csv("data.csv")
df

# no change to sensitivity environment
df + 5

# doubles the sensitivity
df + df

( df * 5, df * df)
```

```
>>> Sensitive(<'DataFrame'>, data.csv ↦ 1, L∞)

>>> Sensitive(<'DataFrame'>, data.csv ↦ 1, L∞)

>>> Sensitive(<'DataFrame'>, data.csv ↦ 2, L∞)

>>> (Sensitive(<'DataFrame'>, data.csv ↦ 5, L∞),Sensitive(<'DataFrame'>, data.csv ↦ ∞, L∞))
```

Side-effects/Mutation

- Since DDUO attaches sensitivity environments to **values** (**instead of variables**), the use of side effects does not affect the soundness of the analysis

- When a program variable is updated to reference a new value, that value's **sensitivity environment remains attached**.

- more capable than traditional type-based static analysis, due to the additional challenges there (e.g. aliasing).

```
total = 0
for i in range(20):
  total = total + df.shape[0]
return total
```

```
>>> Sensitive(<'DataFrame'>,
data.csv ↦ 20, L∞)
```

Conditionals

```
if df.shape[0] == 10:
  return df.shape[0]
else:
  return df.shape[0] * 10000

if dduo.gauss(ε=1.0, δ=1e-5, x) > 5:
  print(dduo.gauss(ε=1.0, δ=1e-5, y))
else:
  print(dduo.gauss(ε=100000000000.0, δ=1e-5, y))
```

- Branching on sensitive values is disallowed

- We don’t want this anyway

- Static Analysis: Take the max!

- Dynamic Analysis: ...?

- **Adaptive** privacy analysis requires use of privacy odometers/filters

Privacy Analysis: Filters and Odometers

```
dduo.laplace(df.shape[0], ε=1.0)

with dduo.EpsOdometer() as odo:
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  _ = dduo.laplace(df.shape[0], ε = 1.0)
  print(odo)

with dduo.EdFilter(ε = 1.0, δ = 10e-6) as odo:
  print('1:', dduo.gauss(df.shape[0], ε=1.0, δ=10e-6))
  print('2:', dduo.gauss(df.shape[0], ε=1.0, δ=10e-6))
```

>>> *9.963971319623278*

>>> *Odometer_ε(*`data.csv ↦ 2.0`*)*

>>> *1: 10.5627*

Traceback (most recent call last):

...

`dduo.PrivacyFilterException`

Loops, Composition, Variants

```
# sequential composition
with dduo.EpsOdometer() as odo:
  for i in range(20):
    dduo.laplace(df.shape[0], ε = 1.0)
  print(odo)

# advanced composition
with dduo.AdvEdOdometer() as odo:
  for i in range(20):
    dduo.gauss(df.shape[0], ε = 0.01, δ = 0.001)

# variant mixing
with dduo.EdOdometer(max_delta = 1e-4) as odo:
  with dduo.RenyiDP(1e-5):
    for x in range(200):
      noisy_count = dduo.renyi_gauss(α = 10,
        ε=0.2, df.shape[0])
  print(odo)
```

- sequential composition

- advanced composition

- variant mixing

Gradient Descent in DDUO

- Bound sensitivity of gradient calculation.
- Add random noise.
- Descend model with noisy gradient.

```python
def dp_gradient_descent(iterations, alpha, eps):
    eps_i = eps/iterations
    theta = np.zeros(X_train.shape[1])
    noisy_count = duet.renyi_gauss(X_train.shape[0], α = alpha, ε = eps)
    for i in range(iterations):
        grad_sum        = gradient_sum(theta, X_train, y_train, sensitivity)
        noisy_grad_sum  = gaussian_mech_vec(grad_sum, alpha, eps_i)
        noisy_avg_grad  = noisy_grad_sum / noisy_count
        theta           = np.subtract(theta, noisy_avg_grad)
    return theta
```

Theorem (Metric Preservation)

If: $\rho_1 \sim^{\Sigma}_{n} \rho_2$
And: $\sigma_1 \sim^{\Sigma}_{n} \sigma_2$
Then: $\rho_1, \sigma_1, e \sim^{\Sigma}_{n} \rho_2, \sigma_2, e$

That is, either $n = 0$ *or* $n = n' + 1$ *and...*

If: $n_1 \leq n$
And: $\rho_1 \vdash \ulcorner \sigma_1, e \urcorner \Downarrow_{n_1} \ulcorner \sigma'_1, v_1 \urcorner$
And: $\rho_2 \vdash \ulcorner \sigma_2, e \urcorner \Downarrow_{n_2} \ulcorner \sigma'_2, v_2 \urcorner$
Then: $n_1 = n_2$
And: $\sigma'_1 \sim^{\Sigma}_{n-n_1} \sigma'_2$
And: $v_1 \sim^{\Sigma}_{n-n_1} v_2$

- the true sensitivity of a program is **guaranteed to be equal to or less than** the sensitivity reported by DDUO's dynamic monitor.
- accurate — even for inputs which **differ entirely from those used** in the dynamic analysis!

Case Studies: Dynamic Enforcement of Privacy

Algorithm	**Libraries Used**	**Baseline**	**Instrumented Version**	**Overhead (% increase)**
Noisy Gradient Descent	NumPy	5.922s	6.302s	6.42%
Multiplicative Weights (MWEM)	Pandas	0.725s	0.833s	14.90%
Private Naive Bayes Classification	DiffPrivLib	2.155s	2.423s	12.44%
Private Logistic Regression	DiffPrivLib	2.022s	3.161s	56.33%

Solo:

Lightweight Static Analysis for Differential Privacy

(Published at OOPSLA 2022)

Why Solo?

+ **Linear types** provide a strategy rooted in type theory and linear logic for tracking resources throughout the semantics of a core lambda calculus.

+ However, **not commonly available** in mainstream programming languages, and when available are usually **not adequately sophisticated**.

What is Solo?

- Static type checking for differential privacy in Haskell
- Doesn’t use linear types!

- Static analysis via (**in tandem** with) Haskell typechecker
- Regular **Haskell type errors** when privacy is violated
- **Privacy cost** along with types of functions & values
- **Encapsulation** of sensitive values via constructor hiding
- **Automatic** type inference

How does it work?

+ Without linear types, where do we attach sensitivities?
+ Previous dynamic sensitivity analyses have attached sensitivities to values.
+ We embed sensitivities in **base types**—the static equivalent of the dynamic strategy of attaching sensitivities to values.

+ **Type-level parameters** to represent sensitivities symbolically.
+ **Type-level computation** to compute symbolic sensitivity expressions.
+ **Parametric polymorphism** to generalize types over sensitivity parameters.
+ Possible in any language with these e.g. Scala, OCaml and Haskell.

Departure from Linear types

+ **case statements** are *restricted* to have the same sensitivity (or privacy cost) in each `case` alternative, rather than max. Solved by `weaken` operation.

+ No verified/typechecked **map or `loop`**. However trusted primitives are available.

+ No **general recursive datatypes**. However, sensitive collection data types are available.

+ In practice, **none of these are typically barriers** to writing differentially private programs.

Solo gets its name from …

Basics: Tracking Sensitivity & Privacy

- **Data source** based approach
- **Type-level sensitivity parameters** on “sensitive” types
- Parametric effect monad: unifies **privacy effect system** with a monadic-style semantics

```haskell
readDoubleFromIO :: ∀ m o. IO (SDouble m '[ '(o, 1) ])

(<+>) :: SDouble 'Diff senv1
  -> SDouble 'Diff senv2
  -> SDouble 'Diff (Plus senv1 senv2)

dbl :: SDouble 'Diff senv -> SDouble 'Diff (Plus senv senv)
dbl x = x <+> x

simplePrivacyFunction :: SDouble 'Diff '[ '(o, 1) ]
  -> EpsPrivacyMonad '[ '(o, 2) ] Double
simplePrivacyFunction x = laplace @2 Proxy $ dbl x
```

Functions

Function types are polymorphic over sensitivity environments

```
-- An s-sensitive function
s_sensitive :: SDouble senv m -> SDouble (ScaleSens senv s) m
-- A 1-sensitive function
one_sensitive :: SDouble senv m -> SDouble senv m
-- map
map :: ∀ m s s₁ a b. (∀ s'. a s' -> b (s * s'))
  -> SList m a s₁
  -> SList m b (s * s₁)
```

Privacy Monad

```haskell
-- values for ε
data EpsPrivacyCost = InfEps | EpsCost TLRat
-- privacy environments, ε-differential privacy
type EpsPrivEnv = [(Source, EpsPrivacyCost)]

return :: a -> EpsPrivacyMonad '[] a
(>>=) :: EpsPrivacyMonad p1 a
  -> (a -> EpsPrivacyMonad p2 b)
  -> EpsPrivacyMonad (EpsSeqComp p1 p2) b
laplace :: Proxy ε
  -> SDouble s Diff
  -> EpsPrivacyMonad (TruncateSens ε s) Double
listLaplace :: Proxy ε
  -> L1List (SDouble Diff) s
  -> EpsPrivacyMonad (TruncateSens ε s) [Double]

addNoiseTwice :: TL.KnownNat (MaxSens s) =>
  SDouble s Diff -> EpsPrivacyMonad '[ '(o,5) ] Double
addNoiseTwice x = do
  y1 <- laplace @2 Proxy x
  y2 <- laplace @3 Proxy x
  return $ y1 + y2
```

- The return operation accepts some value and embeds it in the `PrivacyMonad` without causing any side-effects.
- The `(>>=)` (`bind`) operation allows us to sequence private computations using differential privacy's sequential composition property

Case study: gradient descent

```
-- gradient descent algorithm
gd :: NatS k
  -> NatS ε
  -> SMatrix σ LInf m n SDouble
  -> SMatrix σ LInf m 1 SDouble
  -> EpsPrivacyMonad (ScalePriv k (TruncateSens ε σ)) (Matrix 1 n Double)
gd k t xs ys = do
  let m₀ = matrix (sn32 @ 1) (sn32 @ n) $ \ i j -> 0
      cxs = mclip xs (natS @ 1)
  let f :: SMatrix σ₁ LInf 1 n SDouble
    -> EpsPrivacyMonad (TruncateSens ε σ₁) (Matrix 1 n Double)
      f = \θ -> let g = mlaplace @ε Proxy (natS @5) $ xgradient θ cxs ys
      in msubM (return θ) g
      z = mloop @(TruncateNat t 1) k (sourceM $ xbp m₀) f
  z
```

Proof: similar to DDuo but with type info

Theorem (Metric Preservation)

If: $\gamma_1 \sim \gamma_2 \in \mathcal{G}_n^{\Sigma}[\![\Gamma]\!]$
And: $\Gamma \vdash e : \tau @_m^{\Sigma_1}$
Then: $\gamma_1, e \sim \gamma_2, e \in \mathcal{E}_n^{\Sigma \cdot \Sigma_1}[\![\tau]\!]$

That is, either $n = 0$, *or* $n = n' + 1$ *and...*

If: $n'' \leq n$
And: $\gamma_1 \vdash e \Downarrow_{n''} v_1$
Then: $\exists! v_2.\ \gamma_2 \vdash e \Downarrow_{n''} v_2$
And: $v_1 \sim v_2 \in \mathcal{V}_{n-n''}^{\Sigma \cdot \Sigma_1}[\![\tau]\!]$

Solo: Case Studies

Solo reference implementation is a (~600 loc) Haskell library

- K-means clustering
- Cumulative Distribution Function
- Gradient Descent
- Multiplicative-Weights Exponential Mechanism (MWEM)

Related Work

Related Work: Statically Typed

Linear Type Systems

- Fuzz, DFuzz (only ε-differential privacy)

Indexed Monadic Types

- HOARe^2 (lacks multi-argument support)

Relational Type Systems

- LightDP (lacks sensitivity analysis)

Type Systems Enriched With Program Logics

- Fuzzi (less support for higher-order/type-checking automation)

Related Work: Dynamically Typed

- PINQ (lacks support for general-purpose programming)
- ProPer (per-user budget)
- UniTrax (per-user w/abstract db)
- Testing Methods (counter-example search)

Related Work: Other Privacy Analysis Software Libraries

- DiffPrivLib (no language-based sensitivity/composition)
- Google's Privacy Lib (no language-based sensitivity/composition)
- Ektelo (plans over library of operators, no general PL model)
- DPella (AST analysis, symbolic interpretation)

Future Work & Conclusions

Open Problem/Future Work

- Property-based testing, sensitivity analysis for large software libraries
- Gradual Differential Privacy

Takeaways!

- Analysis for differential privacy is important: buggy programs silently violate your privacy
- Automated enforcement of privacy can be practical in any context!

https://github.com/uvm-plaid/duet

https://github.com/uvm-plaid/dduo-python

https://github.com/uvm-plaid/solo-haskell

Papers: https://chikeabuah.github.io/

Thank You!

Questions?