Designing Programming Languages for Provably Secure Systems

Danfeng Zhang

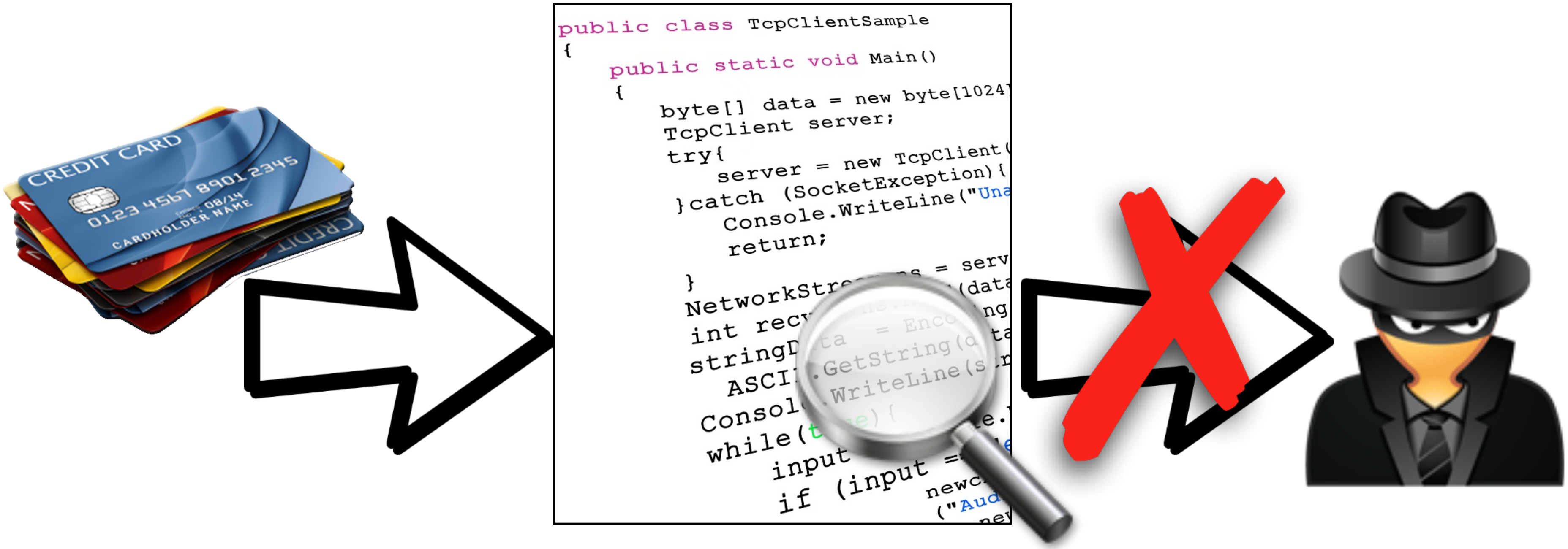
INSR Industry Day 2017

PennState College of Engineering | INSTITUTE FOR NETWORKING AND SECURITY RESEARCH

Need for stronger security

Standard security mechanisms are unsatisfactory

Language-based security



Redesign programming languages for security

Provably enforce security at the language level

Today's talk

Full-system timing channel control
[CCS'10, CCS'11, PLDI'12, ASPLOS'15, ASPLOS'17]

Proving differential privacy
[POPL'17]

Joint work with Aslan Askarov, Andrew Ferraiuolo, Daniel Kifer, Andrew Myers, G. Edward Suh and Yao Wang and Rui Xu

Timing channels

- Information channels in which adversary learns secret data by analyzing timing of public events

Timing channels are real threats to security!

1996 Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems [Kocher]

2003 **Remote** Timing Attacks are Practical [Brumley&Boneh]

2005 **Cache** Attacks and Countermeasures: the Case of AES [Osvik et al.]
Cache Missing for Fun and Profit [Percival]
Cache-Timing Attacks on AES [Bernstein]

2006 Covert and Side Channels Due to **Processor Architecture** [Wang&Lee]

2007 Yet Another MicroArchitectural Attack: Exploiting **I-Cache** [Aciiçmez]
On the Power of Simple **Branch Prediction** Analysis [Aciiçmez et al.]

2009 Hey, You, Get Off of My **Cloud**: Exploring Information Leakage in Third-Party Compute Clouds [Ristenpart et al.]

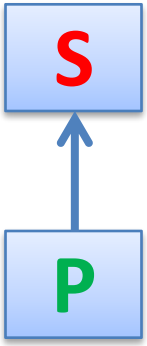
2012 **Cross-VM** Side Channels and Their Use to Extract Private Keys [Y. Zhang et al.]

How to build secure systems that ***provably*** control ***all*** timing channels?

Security model

- Security policy lattice
  - Information has ***label*** describing intended conf.
  - In general, the labels form a *lattice*
  - For this talk, a simple lattice:

    S: secret **P**: public



- Attacker model (at label **P** in the talk)
  - Sees contents of public memory (storage channel)
  - Sees timing of updates to public memory (timing channel)

A subtle example

```
if (secret1)
   secret2:=public1;
else
   secret2:=public2;
public3:=public1;
```

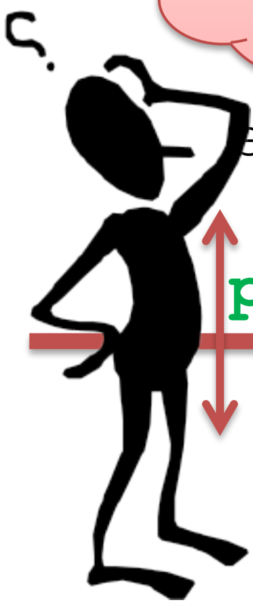
The data cache affects timing

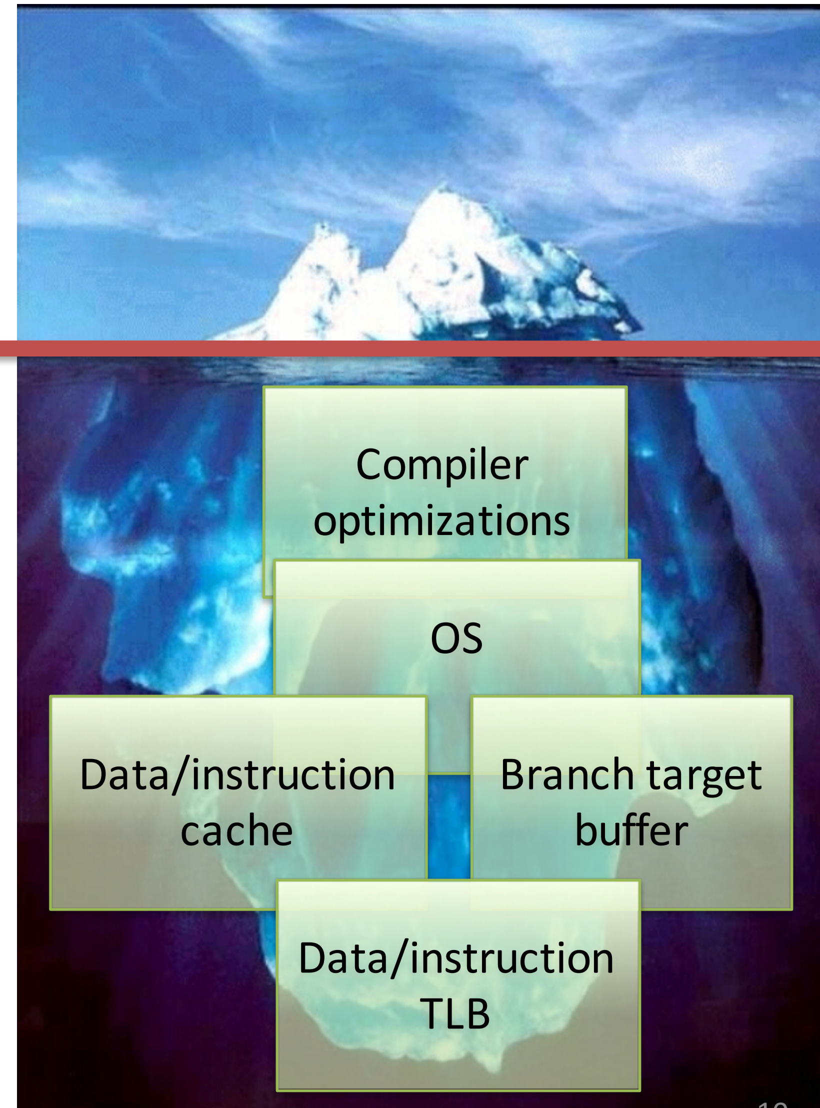
Programming model does not capture timing!

concise and **sufficient** interface?

P… language abstraction

```
…:=pulbic1;
els…
   secret2:=public2;
public3:=public1;
```





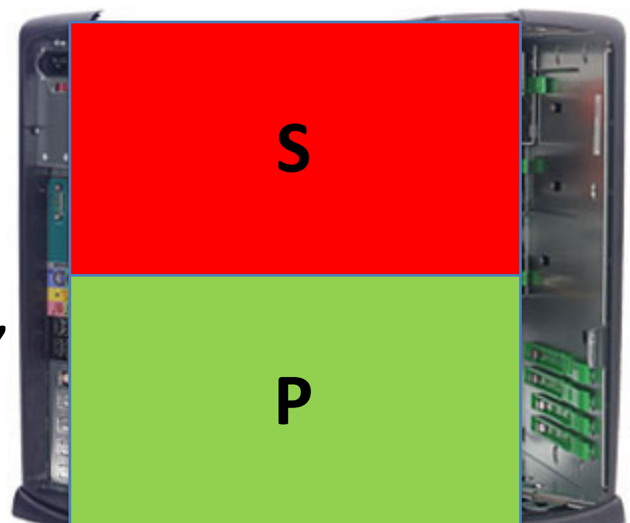
A language-level abstraction [PLDI’12]

Each operation has *read label, write label* governing interaction with machine environment

$$(x := e)_{[\ell_r, \ell_w]}$$

Machine environment: state affecting timing but invisible at language level

machine environment



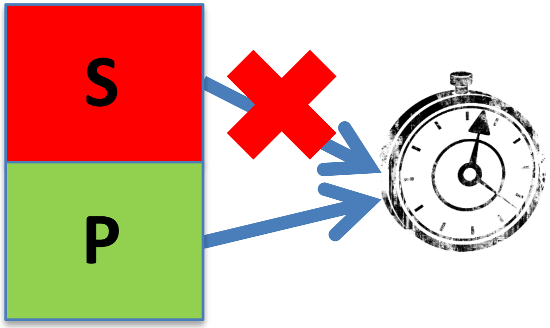
logically partitioned by security label (e.g. public part vs. secret part of cache, time-multiplexed pipeline)

Read labels

$(x := e)_{[\ell_r, \ell_w]}$

- Restricts how machine environment affects timing
- ***Upper bound*** on timing influence
  - e.g., secret cache cannot affect execution time when read label is **P**

$(x := e)_{[\mathbf{P}, \ell_w]}$

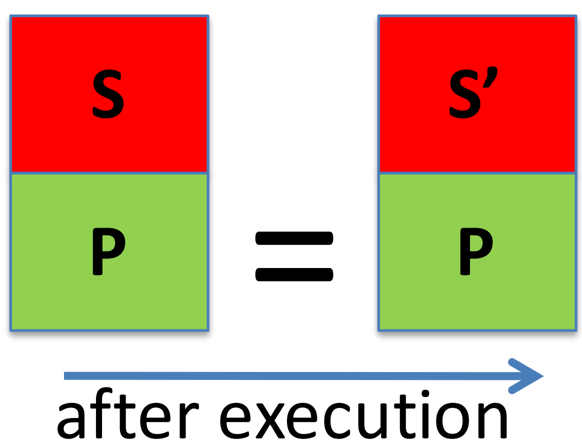


Write labels

$(x := e)_{[\ell_r, \ell_w]}$

- Restricts how machine environment is modified
- ***Lower bound*** on updates to machine env.
  - e.g., no updates to public cache when write label is **S**

$(x := e)_{[\ell_r, \mathbf{S}]}$



A core language with read/write labels

$$e ::= n \mid x \mid e\ op\ e$$

$$c ::= \texttt{skip}_{[\ell_r,\ell_w]} \mid (x := e)_{[\ell_r,\ell_w]} \mid c;c \mid (\texttt{while}\ e\ \texttt{do}\ c)_{[\ell_r,\ell_w]}$$
$$\mid (\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2)_{[\ell_r,\ell_w]} \mid (\texttt{sleep}\ e)_{[\ell_r,\ell_w]}$$

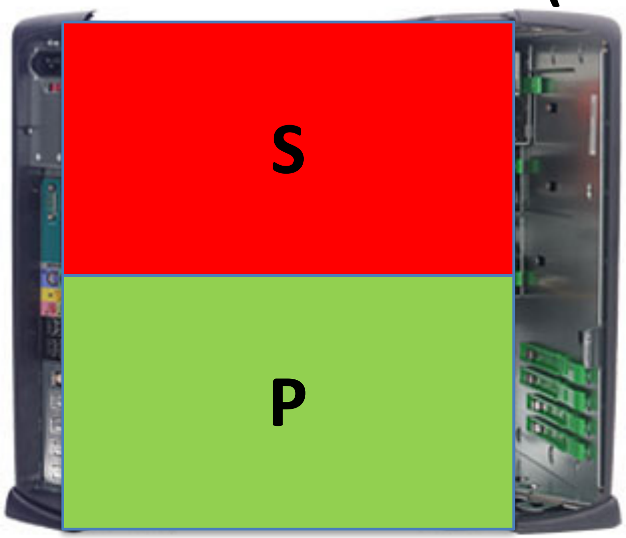
Most can be automatically inferred

Read/Write labels form a contract

$_{[\ell_r,\ell_w]}(x := e)$

Reason about timing channels based on the contract

machine environment (ME)



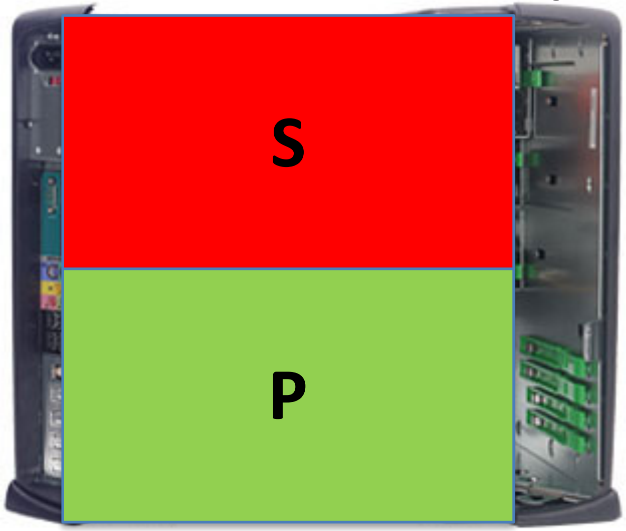
Obeys the timing contract (formalized in [PLDI'12])

Security enforcement

$_{[\ell_r,\ell_w]}(x := e)$

A type system checks timing channels [PLDI'12]

machine environment (ME)



A Verilog extension that statically verifies HW designs [ASPLOS'15, 17]

Formally verified MIPS processor

Rich ISA: runs OpenSSL with off-the-shelf GCC

Classic 5-stage in-order pipeline

- Typical pipelining techniques
  - data hazard detection
  - stalling
  - data bypassing

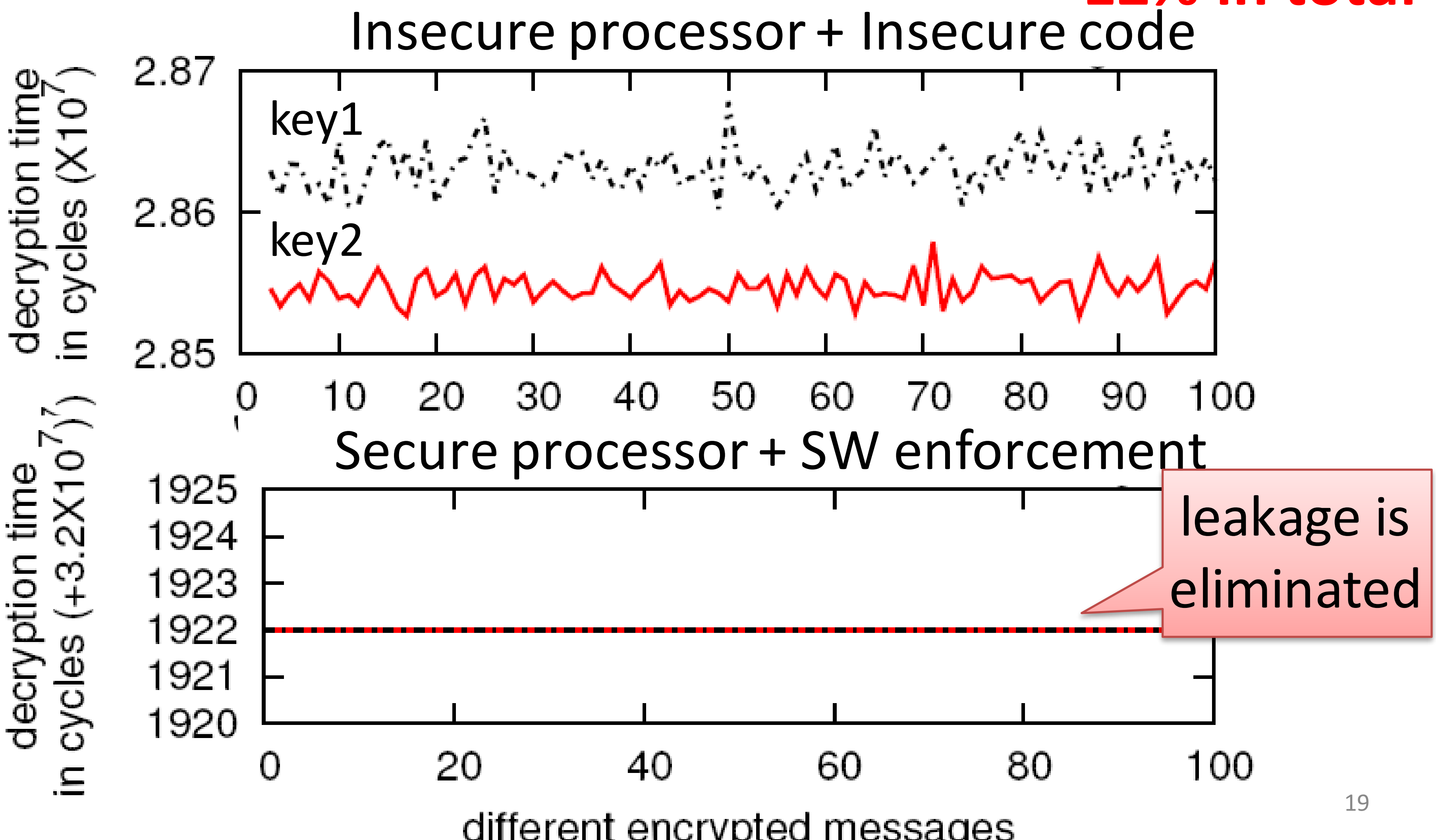
Overhead of hardware resources

unmodified/ insecure

	Baseline	Verified	Overhead
Delay w/ FPU (ns)	4.20	4.20	**0%**
Delay w/o FPU (ns)	1.64	1.66	**1.21%**
Area (μm^2)	399400	402079	**0.67%**
Power (mW)	575.5	575.6	**0.02%**

RSA case study

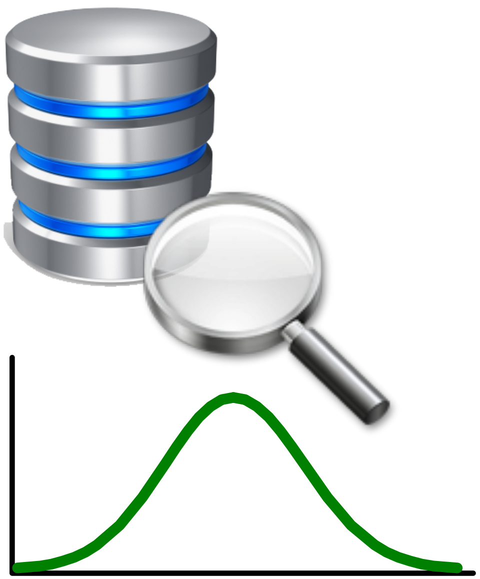
Overhead: ~12% in total



Today’s talk

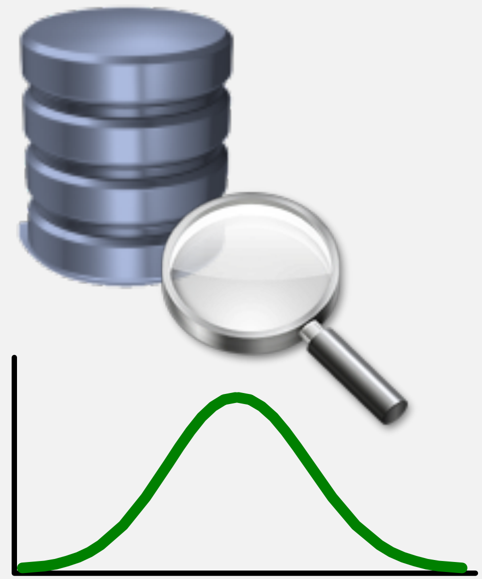
Proving differential privacy
[POPL’17]

Database w/
Alice's data



μ_1

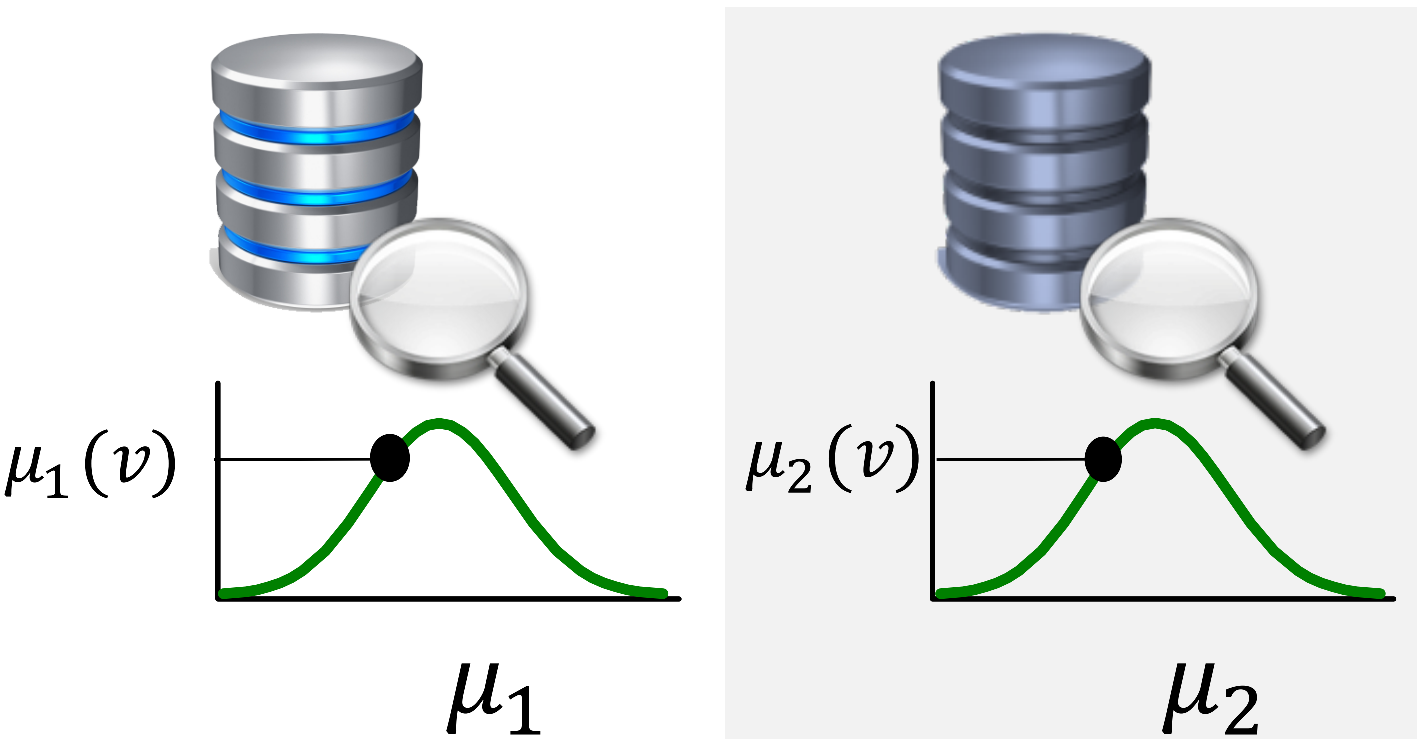
Database ***w/o***
Alice's data



μ_2

Alice's data remain private if μ_1, μ_2 are *close*

(Pure) Differential privacy



If for any *adjacent* databases and value v, $\mu_1(v)/\mu_2(v) \leq e^{\epsilon}$ for some constant ϵ, then a computation is ϵ-private

Privacy Cost

Motivation

DP has seen explosive growth since 2006

– U.S. Census Bureau [Machanavajjhala et al. 2008]

– Google Chrome Browser [Erlingsson et al. 2014]

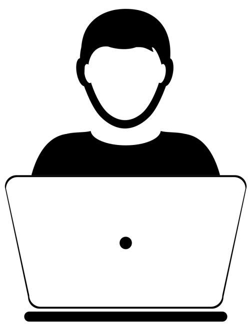
– Apple's new data collection efforts [Greenberg 2016]

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

LightDP: Overview



Source Program

```
η1 := Lap (2/ε);
T̃ := T + η1;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  η2 := Lap (4N/ε);
  if (q[i] + η2 ≥ T̃) then
    out:= true::out;
    c1 := c1 + 1;
  else
    out:= false::out;
    c2 := c2 + 1;
  i := i+1;
```

Relational, Dependent Type System

Target Program with distinguished variable $\mathbf{v}_\epsilon$

```
vε := 0;
havoc η1; vε := vε + ε/2;
T̃ := T + η1;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  havoc η2; vε := vε + (q[i] + η2 ≥ T̃?2 : 0) × ε/4N;
  if (q[i] + η2 ≥ T̃) then
    out:= true::out;
    c1 := c1+1;
  else
    out:= false::out;
    c2 := c2+1;
  i := i+1;
```

Main Theorem

Source program type checks ✓ + $\mathbf{v}_\epsilon$ bounded by constant ϵ in the target program ✓

➡ Source program is ϵ-private ✓

Source language: syntax

Random variable

Random Expression (e.g., Laplace dist.)

$$\begin{aligned}\text{Commands} \quad c \quad ::= \;& \texttt{skip} \mid x := e \mid \eta := g \mid c_1; c_2 \mid \texttt{return}\ e \mid \\ & \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid \texttt{while}\ e\ \texttt{do}\ c\end{aligned}$$

Relational types

$$\mathcal{B}_{d}$$

Base Type

Distance

Example

$\Gamma(x): \mathrm{num}_0$

$\Gamma(y): \mathrm{num}_1$

e.g., int, real

Related Memories

x: u	x: u
y: v	y: v+1

Dependent types

$\mathcal{B}_{\mathrm{d}}$ Can be a program variable

Example

$\Gamma(x)\colon \mathrm{num}_0$

$\Gamma(y)\colon \mathrm{num}_x$

Related Memories

x: u	x: u
y: v	y: v + u

Dependent types

$\mathcal{B}_{\text{d}}$

Can be a non-prob. expression

Example

$\Gamma(x): \mathrm{num}_0$

$\Gamma(y): \mathrm{num}_{x \geq 1 ? 2 : 0}$

Related Memories

x: u

y: v

x: u

$y: \begin{cases} \mathrm{v} + 2, \mathrm{u} \geq 1 \\ \mathrm{v} \quad\quad , \mathrm{u} < 1 \end{cases}$

Notation

$m_1 \; \Gamma \; m_2$ if m_1 and m_2 are related by Γ

(for the non-probabilistic subset)

Types form an invariant on two related program executions:

If initial memories m_1 Γ m_2

Then after executing
a well-typed program,
final memories m_1' Γ m_2'

Enforced by a type system

In general, maintaining the distances may incur privacy cost

Source program

```
η1 := Lap (2/ε);
T̃ := T + η1;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  η2 := Lap (4N/ε);
  if (q[i] + η2 ≥ T̃) then
    out:= true::out;
    c1 := c1 + 1;
  else
    out:= false::out;
    c2 := c2 + 1;
  i := i+1;
```

Type System

Target program with distinguished variable $\mathbf{v}_\epsilon$

```
v_ε := 0;
havoc η1; v_ε := v_ε + ε/2;
T̃ := T + η1;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  havoc η2; v_ε := v_ε + (q[i] + η2 ≥ T̃?2 : 0) × ε/4N;
  if (q[i] + η2 ≥ T̃) then
    out:= true::out;
    c1 := c1+1;
  else
    out:= false::out;
    c2 := c2+1;
  i := i+1;
```

$$\Gamma \vdash c \rightharpoonup c'$$

source program

target program

Target language

set x to arbitrary value

havoc x

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

Verification task in the target language:
Proving $\mathbf{v}_\epsilon$ is bounded by some constant ϵ in any execution (in a non-probabilistic 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

```
η1 := Lap (2/ϵ);
T̃ := T + η1;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  η2 := Lap (4N/ϵ);
  if (q[i] + η2 ≥ T̃) then
    out:= true::out;
    c1 := c1 + 1;
  else
    out:= false::out;
    c2 := c2 + 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

$\mathbf{c1, c2, i : num}_0; \tilde{T}, \eta_1 : \mathbf{num}_1; \eta_2 : \mathbf{num}_{q[i]+\eta_2 \geq \tilde{T}?2:0}$

η_1 := **Lap** $(2/\epsilon)$;
$\tilde{T}$:= $T + \eta_1$;
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
  η_2 := **Lap** $(4N/\epsilon)$;
  if $(q[i] + \eta_2 \geq \tilde{T})$ **then**
    out:= true::out;
    c1 := c1 + 1;
  else
    out:= false::out;
    c2 := c2 + 1;
  i := i+1;

Distance depends on the value of ith query answer ($q[i]$)

Type Inference

Types can be inferred by the inference algorithm of LightDP

Target program

$\mathbf{v}_\epsilon := 0;$

havoc $\eta_1; \mathbf{v}_\epsilon := \mathbf{v}_\epsilon + \epsilon/2;$

~~η_1 := Lap $(2/\epsilon)$;~~

$\tilde{T}$:= $T + \eta_1$;

c1 := 0; c2 := 0; i := 0;

while (c1 < N)

  ~~η_2 := Lap $(4N/\epsilon)$;~~

  havoc $\eta_2; \mathbf{v}_\epsilon := \mathbf{v}_\epsilon + (q[i] + \eta_2 \geq T?2 : 0) \times \epsilon/4N;$

  if ($q[i] + \eta_2 \geq \tilde{T}$) **then**

    out:= true::out;

    c1 := c1 + 1;

  else

    out:= false::out;

    c2 := c2 + 1;

  i := i+1;

Completing the proof

$\mathbf{v}_\epsilon := 0;$

havoc $\eta_1; \mathbf{v}_\epsilon := \mathbf{v}_\epsilon + \epsilon/2;$

$\tilde{T} := T + \eta_1;$

```
c1 := 0; c2 := 0; i := 0;
while (c1 < N)
```

$\mathtt{Invariant} : \mathtt{c1} \leq N \wedge \mathbf{v}_\epsilon = \epsilon/2 + \mathtt{c1} \times \frac{\epsilon}{2N}$

havoc $\eta_2; \mathbf{v}_\epsilon := \mathbf{v}_\epsilon + (q[i] + \eta_2 \geq \tilde{T} ? 2 : 0) \times \epsilon/4N;$

if $(q[i] + \eta_2 \geq \tilde{T})$ then

```
    out:= true::out;
    c1 := c1+1;
  else
    out:= false::out;
    c2 := c2+1;
  i := i+1;
```

Loop Invariant

Postcondition: $\mathbf{v}_\epsilon \leq \epsilon$

Main Theorem

Source program type checks
+ $\mathbf{V}_\epsilon$ bounded by constant ϵ
= source program is ϵ-private

Thank you!