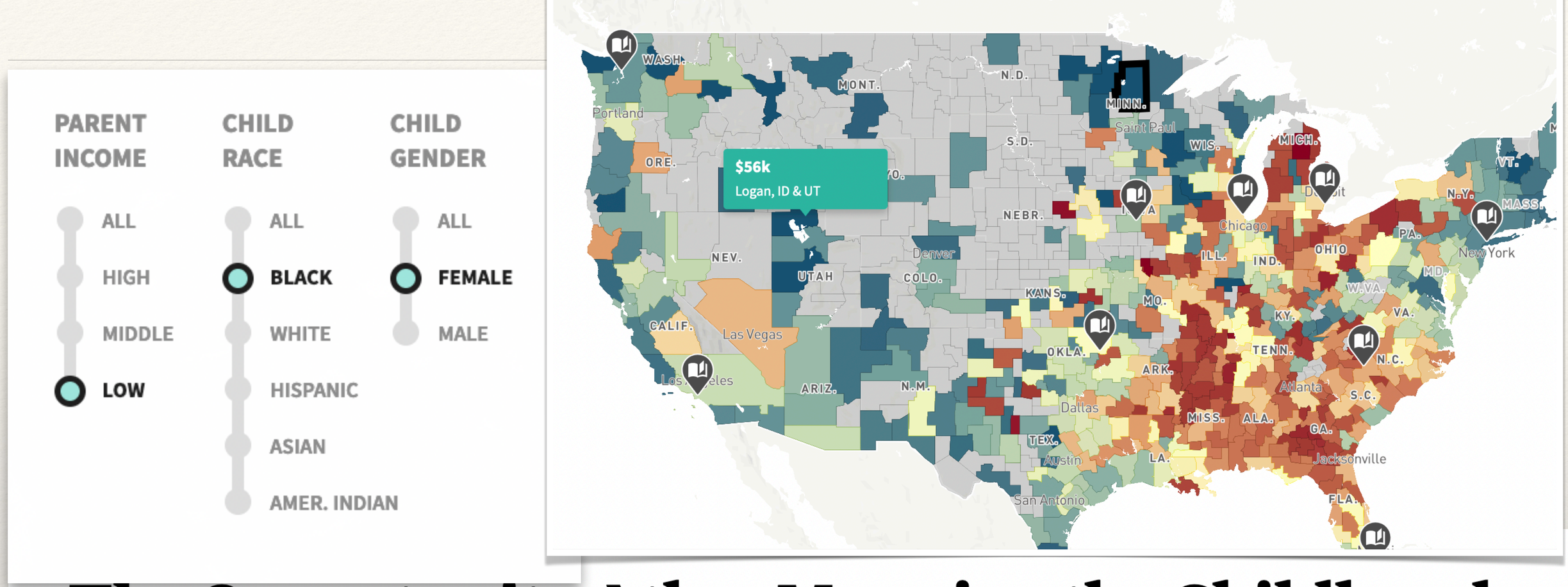
Towards Leakage-Resistant Machine Learning in Trusted Execution Environments (ongoing work)

Mukesh Tiwari, University of Cambridge (work was done at the University of Melbourne)



The Opportunity Atlas: Mapping the Childhood Roots of Social Mobility

Raj Chetty, John N. Friedman, Nathaniel Hendren,
Maggie R. Jones & Sonya R. Porter

Our Goal: End-to-End Secrecy

- No data identification, after the processing

Membership Inference Attacks Against Machine Learning Models

Reza Shokri
Cornell Tech
shokri@cornell.edu

Marco Stronati*
INRIA
marco@stronati.org

Congzheng Song
Cornell
cs2296@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

Differential Privacy

Our Goal: End-to-End Secrecy

- No secret data leak, during the processing

Trusted Execution Environment

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.

Initialize θ_0 randomly

for $t \in [T]$ **do**

Take a random sample L_t with sampling probability L/N

Compute gradient

For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$

Clip gradient

$\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$

Add noise

$\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$

Descent

$\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$

Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

Deep Learning with Differential Privacy

October 25, 2016

Martín Abadi* Andy Chu* Ian Goodfellow†
H. Brendan McMahan* Ilya Mironov* Kunal Talwar*
Li Zhang*

Side Channel

Most traditional side channels—regardless of technique—can be mitigated by applying all three of the following general “constant time”[2] principles, listed here at a high level. Software that handles secrets must follow these principles to guard against side channel methods. We discuss details and examples of these principles later.

1. Ensure runtime is independent of secret values.
2. Ensure code access patterns[3] are independent of secret values.
3. Ensure data access patterns[4] are independent of secret values.

These principles are conceptually simple, but sometimes can be difficult to implement in practice, depending on the complexity of your algorithms. Luckily, most developers won’t need to work on the most complex situations.

Solution: Information Flow for C

SecCSL: Security Concurrent Separation Logic

Authors | Authors and affiliations

Gidon Ernst ✉, Toby Murray

$$\exists c\ d.\ \texttt{rec} \mapsto (c, d) \wedge c :: \texttt{low} \wedge d :: (c\ ?\ \texttt{high} : \texttt{low})$$

Formally Verified in Isabelle/HOL

https:covern.org/secc/

Gradient Descent Algorithm

procedure CONSTANT-DPGD($n, T, t, \theta_1^0, \theta_2^0, \gamma, \epsilon$) ▷ number of data points, number of iterations, clipping range, initial (guess) slope, initial (guess) intercept, learning rate, privacy parameter

Data: $\{(x_i, y_i)\}_{i=1}^{n}$

for t := 1 to T **do**

for i := 1 to n **do**

$\tilde{y}_i = \theta_1^t * x_i + \theta_2^t$

$\Delta_{i,t} = \begin{pmatrix} 2(\tilde{y}_i - y_i)x_i \\ 2(\tilde{y}_i - y_i) \end{pmatrix}$

$\Delta_{i,t}^c = \{\Delta_{i,t}\}_{-t}^{t}$

end for

$\Delta_t = \frac{1}{n}\sum_{i=1}^{n} \Delta_{i,t}^c + Lap(0, 4t/\epsilon)$

$[\theta_1^{t+1}, \theta_2^{t+1}] = [\theta_1^t, \theta_2^t] - \gamma * \Delta_t$

end for

return $[\theta_1^{T+1}, \theta_2^{T+1}]$

end procedure

Gradient Clipping

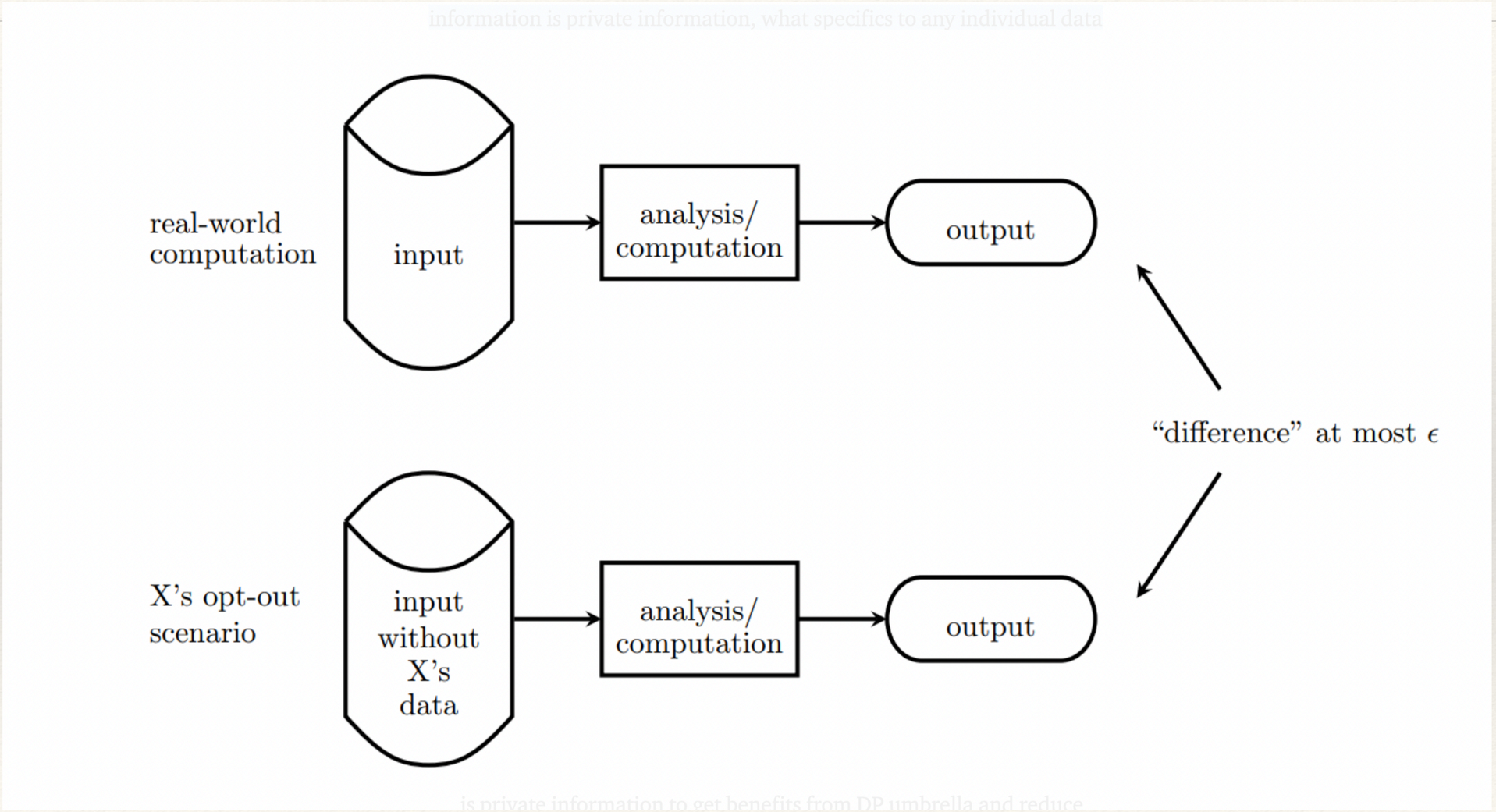
```
>>> import numpy as np
>>> in_array = [1, 2, 3, 4, 5, 6, 7, 8 ]
>>> out_array = np.clip(in_array, 2, 5)
>>> out_array
array([2, 2, 3, 4, 5, 5, 5, 5])
```

```
1  void clip_gradient(struct gradient *gs, int n, double tau)
2  _(requires tau > 0)
3  _(requires valid_gr
4  _(ensures  valid_gr
5  {
6    _(unfold valid_gr
7    _(assert (gs == n
8    if(gs == NULL)
9    {
10     _(assert n == 0)
11     _(fold valid_gradient_
12     return; // reached the
13   }
14   else
15   {
16
17     (exists double u, doub
18         &gs->m |-> u && (u
19         ..................
20
21     double m = gs->m;
22     double c = gs->c;
```

```
37       _(apply bounded_lemma(l, h, kl, kr, m, tau, retm);)
38       // Proof that retm is within [-tau, tau]
39       _(assert -tau <= retm && retm <= tau)
40       gs->m = retm;
41
```

```
54
55       _(apply bounded_lemma(u, d, wl, wr, c, tau, retc);)
56       // Proof that retc is within [-tau, tau]
57       _(assert -tau <= retc && retc <= tau)
58       gs->c = retc;
59
60
61       clip_gradient(gs->next, n-1, tau);
62       _(fold valid_gradient_inv(;gs, n))
63     }
```

Differential Privacy



Secure C: Not Capable of Differential Privacy Reasoning

We capture the program history as IO traces

```
87   /*
72  struct event
73  {
74    event_status est;
75    double m;
76    double c;
77    // When est == Training
78    double m_old; // old m
79    double c_old; // old c
80    double dm; // averaged m
81    double dc; // averaged c
82    double noise1; // noise added to m_new = m_old - learning_rate * (dm + noise1)
83    double noise2; // noise added to c_new = c_old - learning_rate * (dc + noise2)
84  };
85
```

Secure C: Not Capable of Differential Privacy Reasoning

We ensure that during the program execution, the IO trace is well-formed

```
185  _(predicate safe_to_release_final_m_c(double noism, double noisc, double eps, double refvalue, double lrate,
186        list<struct event> ios, struct event e, list<struct event> rs, struct event f, int k)
187        ((iost == append(ios, cons(e, append (rs, cons(f, nil)))) && // split the whole trace into previous ++
188          (count_total_budget(eps, refvalue, ios) >= k * eps) && // we have enough budget to train k iteration
189          e.est == Received && // received the parameters from server
190          (all_training_trace(rs) <=> true) && // everything in rs is training
191          (well_formed_training_trace(lrate, rs) <=> true) && // well formed training trace
192          (length(rs) == k) && // k training history
193          f.est == Conveyed &&  f.m == noism && f.c == noisc && // sent the trained parameters to server
194          (count_total_budget(eps, refvalue, iost) >= (count_total_budget(eps, refvalue, ios) - k * eps)) )))
```

Secure C: Not Capable of Differential Privacy Reasoning

Finally, we ensure that well-formed condition holds when returning the values

```
1538        _(unfold safe_to_release_final_m_c(m_old, c_old, eps, refvalue, learn_rate, iost2, ios, e, iotl, et1, k))
1539
1540
1541        return Success;
1542    }
```

Recap: Differential Privacy

We record data, as program history

We define constraints on the data, i.e., well-formed condition (predicate)

We ensure that the constraints —well-formed condition (predicate)— hold when function returns the relevant values

Experiments

Synthetic Data (x, y), generated according to $y = m * x + c$ for some given m and c

Result on synthetic data are encouraging, but it requires more experiments on real world data

Learnings

IO traces are too much low level, so if you want to prove some new property about your implementation you need to think if have captured the right data or not.

Thank You