... thanks ...

I take it that my job today is to give you some feeling for what a programming language perspective on DP might be.

As a PL person, I come to the area of DP almost entirely as an outsider: the algorithmic techniques and underlying mathematics are very different from what I use in my day to day life. But I do recognize one thing: there are a lot of programs here, a lot of proofs about those programs, and a lot of common structure to those proofs.

Let me illustrate with a little story.
In the middle ages, written communication was the sole province of a handful of experts. Writing was hard and every instance of a document had to be laboriously copied out by hand.
Let’s look a little more closely at what this man is writing. He’s just written out a proof that his favorite query is differentially private. But you can see the expression on his face is a bit grim. This is because he knows that tomorrow his boss is going to suggest a slight variation of the query, and he’s going to have to write out his proof all over again.

And he’s not the only person that’s unhappy.
This is the owner of the database. She’s angry because she has just spent all day checking the poor monk’s previous proof and convincing herself that it’s safe to run it against her very secret database.

And it’s not just that! Besides this query...
... she’s got stacks of other queries from other monks that she also needs to check!

So clearly, we’ve got a fundamental scaling problem here. We need to do something to help all these people not to have to work so hard to write their queries and especially so that the data owner doesn’t have to work so hard to check that they are safe!

And when you put it this way, it’s equally clear that this is a job for...
Programming languages, of course!

That is, we need to find good ways of automating this process.
**PL for DP**

- **Goal:**
  - Provide tools for expressing queries so that their privacy can be *mechanically verified*

- **Tools:**
  - compositionality
  - little languages
  - type systems
  - proof checkers

... different people focus on different points in the design space. Note that point (1) is truly critical, since otherwise the database owner becomes a bottleneck in the whole process. Point (2) is important too, but if we have an approach that makes database queriers have to work a little harder in return for being able to express more interesting queries, that might be a good tradeoff.

...
here’s a map of the territory we’re going to cover
I’ve tried to cover a lot of ground -- almost certainly more than I can get to in an hour -- but I’ve tried to front-load some of the most interesting parts, and if I end up skimming some things at the end at least they’ll be here in the slides for people to refer to later if they want.
Any questions?
Dynamic tracking
... how familiar is it in this community? (How many people here feel they have a good understanding of how PINQ works?)
Privacy INtegrated Queries

- Common platform for differentially private data analyses.
- Provides interface to data that looks very much like LINQ (C#’s “language-integrated queries”)
- All access through the interface is guaranteed to be differentially private
- (Non-privacy-expert) data analysts write arbitrary LINQ code against data sets in C#.
var data = new PINQueryable<SearchRecord>(...);

var users = from record in data
    where record.Query == argv[0]
    groupby record.IPAddress;

Console.WriteLine(argv[0] + "::" + users.Count(0.1));
How it works...

• Each private data source is wrapped in a PINQueriable object, which is responsible for...
  • mediating accesses to the underlying data
  • remembering how much privacy budget is left
  • deducting from the budget whenever an aggregation operator is applied to this PINQueriable object (or any other one derived from it)
  • denying access once the budget is exhausted
Aggregation operations

- **NoisyCount**
  - arguments: a PINQueriable and a desired accuracy for the count
  - calculates how much privacy budget must be expended to produce an answer with this accuracy
    - asks PINQueriable to deduct this much
  - returns count plus appropriate Laplace noise

- Similarly: NoisySum, NoisyAverage, etc.
Transformations

• Each transformation method...
  • maps a PINQueriable to one or more new PINQueriables...
  • that, when later aggregated, will forward the privacy cost to the original object...
  • after applying an appropriate scale factor (i.e., after taking account of the sensitivity of the transformation).
Transformations

- **Where**: takes a predicate and returns a new PINQueriable wrapping the subset of the data satisfying the predicate.

- **GroupBy**: takes a function mapping records to key values, and results in a list of groups: for each observed key, the group of records that map to that key.

- **Join**: takes two data sets, key selection functions for each, and returns the list of all pairs of elements whose keys match (to prevent blowup in the size of the output, each input data set is first grouped by its join keys, so that each join key becomes a primary key).

- **Partition**: like GroupBy, but must be explicitly provided with a list of candidate keys; its result is a list of PINQueryable objects, one for each candidate key, containing the (possibly empty) subset of records with this key.
Evaluation

• Advantages of the PINQ approach:
  • Simple to implement and explain
  • Flexible: wide range of DP queries can be expressed

• Limitation: No static checking → privacy budget violations only detected at the end
  • may waste time or privacy if a long-running privacy-demanding computation needs more budget than is available
Airavat


(Thanks to Roy for slides!)
Airavat

Framework for privacy-preserving MapReduce computations with untrusted code.

Airavat is the elephant of the clouds (Indian mythology).
Background: MapReduce

map\((k_1, v_1) \rightarrow \text{list}(k_2, v_2)\)
reduce\((k_2, \text{list}(v_2)) \rightarrow \text{list}(v_2)\)

Data 1 → Map phase → Reduce phase → Output

Data 2

Data 3

Data 4
MapReduce example

Map(input) → { if (input has iPad) print (iPad, 1) }
Reduce(key, list(v)) → { print (key + “;” + SUM(v)) }

Counts no. of iPads sold
Split MapReduce into untrusted mapper + trusted reducer

Limited set of stock reducers

MapReduce program for data mining

No need to audit

Untrusted Mapper

Trusted Reducer

Airavat

Data

Data
Programming model

Need to confine the mappers!
Guarantee: Protect the privacy of data providers

MapReduce program for data mining

No need to audit

Untrusted Mapper

Trusted Reducer

Airavat

Data

Data
Airavat mechanisms

Mandatory access control

Prevent leaks through storage channels like network connections, files…

Differential privacy

Prevent leaks through the output of the computation

Data → Map → Reduce → Output
Malicious mappers may output values outside the range. If a mapper produces a value outside the range, it is replaced by a value inside the range. User not notified... otherwise possible information leak.

Ensures that code is not more sensitive than declared.
Static Analysis
Motivation

- Want to know in advance how much privacy a query will use
- Dynamic tracking of privacy depletion gives us no well-grounded way of looking at a program and predicting its privacy cost

(either as an author of the program or as a database owner)

XXXXXX improve this!
Fuzz

Reed and Pierce, Distance makes the types grow stronger: A calculus for differential privacy. ICFP 2010.

See also: Palamidessi and Stronati: Differential Privacy for Relational Algebra: Improving the Sensitivity Bounds via Constraint Systems, QAPL 2012
Fuzz

- Higher-order functional language (ML-like)
- Static type system features
  - sensitivity tracking based on linear logic
  - internalized type of probability distributions
- Differential privacy guaranteed by typechecking
Sensitivity $\rightarrow$ Privacy

+ 

PL for Sensitivity

= 

PL for Privacy
Quick example

Suppose we have the following functions:

\[\text{over\_40} : \text{row} \rightarrow \text{bool}\]
\[\text{size} : \text{db} \rightarrow \mathbb{R}\]
\[\text{filter} : (\text{row} \rightarrow \text{bool}) \rightarrow \text{db} \rightarrow \text{db}\]
\[\text{add\_noise} : \mathbb{R} \rightarrow \mathcal{O}\mathbb{R}\]

This expression computes a differentially private count of database rows satisfying \text{over\_40}:

\[\lambda d : \text{db}. \text{add\_noise} (\text{size} (\text{filter over\_40} d)) : \text{db} \rightarrow \mathcal{O}\mathbb{R}\]
Punchline

Typing guarantees differential privacy

**Theorem** If $e$ is a closed program with $\vdash e : \tau \rightarrow \circ \sigma$

is an $\epsilon$-differentially private function from $\tau$ to $\sigma$. 
In “classic DP,” there are basically only two interesting types: real numbers and databases. We would like to introduce other base types, and type operators to build new types from old ones. To do this, we need to extend the notion of “distance” to these new types.

Now we can straightforwardly generalize the definition of $c$-sensitivity to arbitrary types.
Base type

The primitive type $\mathbb{R}$ has the usual metric:

$$d_\mathbb{R}(x, y) = |x - y|$$
Sets

τ set is a type for each type τ, with Hamming metric

\[ d_{\text{set}}(S_1, S_2) = ||S_1 \triangle S_2|| \]

Primitives:

- \textit{size} : \tau \text{ set} \rightarrow \mathbb{R}
- \textit{setfilter} : (\tau \rightarrow \text{bool}) \rightarrow \tau \text{ set} \rightarrow \tau \text{ set}
- \textit{setmap} : (\sigma \rightarrow \tau) \rightarrow \tau \rightarrow \sigma \text{ set} \rightarrow \sigma \text{ set}
- \cap, \cup, \setminus : \tau \text{ set} \otimes \tau \text{ set} \rightarrow \tau \text{ set}
- \textit{split} : (\tau \rightarrow \text{bool}) \rightarrow \tau \text{ set} \rightarrow \tau \text{ set} \otimes \tau \text{ set}
Scaling

For each type \( \tau \), let \( !, \tau \) be the type with the same values as \( \tau \), but with the metric 'scaled up' by \( r \):

\[
d_{!, \tau}(x, y) = r \cdot d_\tau(x, y)
\]

**Proposition** A function \( f \) is a \( c \)-sensitive function in \( \tau_1 \rightarrow \tau_2 \) iff it is a 1-sensitive function in \( !c \tau_1 \rightarrow \tau_2 \).
Pairs

$\tau_1 \otimes \tau_2$ is the type of pairs $(v_1, v_2)$ with $v_1 \in \tau_1$ and $v_2 \in \tau_2$.

Distance between pairs is the sum of the distances between components:

$$d_{\tau_1 \otimes \tau_2}((v_1, v_2), (v'_1, v'_2)) = d_{\tau_1}(v_1, v'_1) + d_{\tau_2}(v_2, v'_2)$$
Examples

1-sensitive functions in $\mathbb{R} \otimes \mathbb{R} \rightarrow \mathbb{R}$:

$$f_1(x, y) = x + y \quad f_2(x, y) = x - y$$

1-sensitive functions in $\mathbb{R} \otimes \mathbb{R} \rightarrow \mathbb{R} \otimes \mathbb{R}$:

$$f_3(x, y) = (x, y) \quad f_4(x, y) = (y, x)$$

$$f_5(x, y) = (x + y, 0) \quad cswp(x, y) = \begin{cases} (x, y) & \text{if } x < y \\ (y, x) & \text{otherwise} \end{cases}$$

Non-example:

$$f_6(x, y) = (x, x)$$

is not a 1-sensitive function in $\mathbb{R} \otimes \mathbb{R} \rightarrow \mathbb{R} \otimes \mathbb{R}$. 
Another metric for pairs

$\tau_1 \& \tau_2$ consists of pairs $\langle v_1, v_2 \rangle$ with the metric

$$d_{\tau_1 \& \tau_2}(\langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle) = \max(d_{\tau_1}(v_1, v'_1), d_{\tau_2}(v_2, v'_2))$$

E.g.

$$f_7(x, y) = \langle x, x \rangle$$

is a 1-sensitive function $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R} \& \mathbb{R}$

**Proposition** If $f : \tau \to \tau_1$ and $g : \tau \to \tau_2$ are c-sensitive, then $\lambda x. \langle f x, g x \rangle$ is a c-sensitive function in $\tau \to \tau_1 \& \tau_2$.

[& lets us combine outputs of c-sensitive functions even if they depend on common inputs.]
[We have already established that the presence of $!_r$ means that having 1-sensitive functions suffices to express $c$-sensitive functions for all $c$, so we need not specially define an entire family of $c$-sensitive function type constructors:]
Disjoint unions

\( \tau_1 + \tau_2 \) is the disjoint union of \( \tau_1 \) and \( \tau_2 \)

Values: \( \{ \text{inj}_1 v \mid v \in \tau_1 \} \cup \{ \text{inj}_2 v \mid v \in \tau_2 \} \)

Metric:

\[
d_{\tau_1 + \tau_2}(v, v') = \begin{cases} 
  d_{\tau_1}(v_0, v'_0) & \text{if } v = \text{inj}_1 v_0 \text{ and } v' = \text{inj}_1 v'_0; \\
  d_{\tau_2}(v_0, v'_0) & \text{if } v = \text{inj}_2 v_0 \text{ and } v' = \text{inj}_2 v'_0; \\
  \infty & \text{otherwise.}
\end{cases}
\]
**Booleans**

This metric defines an *extremely* disjoint union of two components. E.g., for the type

\[ \text{bool} = 1 + 1 \]

we have

\[
\begin{align*}
    d(\text{true}, \text{true}) &= d(\text{false}, \text{false}) = 0 \\
    d(\text{true}, \text{false}) &= d(\text{false}, \text{true}) = \infty
\end{align*}
\]

Upshot: Easy to write \(c\)-sensitive functions *from* `bool` to other types, but hard for a nontrivial function *to* `bool` to be \(c\)-sensitive. E.g.,

\[ gtzero : \mathbb{R} \rightarrow \text{bool} \]

is not \(c\)-sensitive for any finite \(c\).
Lists

Two lists of different lengths are at distance $\infty$ from each other (this corresponds to the definition of the metric on disjoint union types).

For two lists $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$ of the same length,

$$d_{\text{list}}([x_1, \ldots, x_n], [y_1, \ldots, y_n]) = \sum_{i=1}^{n} |x_i - y_i|.$$
Sorting

Can't have this:

$$\preceq_R : \mathbb{R} \otimes \mathbb{R} \rightarrow \text{bool}$$

So use this:

$$cswp : \mathbb{R} \otimes \mathbb{R} \rightarrow \mathbb{R} \otimes \mathbb{R}$$

Now:

$$insert : \mathbb{R} \rightarrow \mathbb{R} \text{list} \rightarrow \mathbb{R} \text{list}$$

$$insert \ x \ [] = [x]$$

$$insert \ x \ (h :: tl) = \text{let} \ (a, b) = cswp \ (x, h) \ \text{in}$$

$$a :: (insert \ b \ tl)$$

$$sort : \mathbb{R} \text{list} \rightarrow \mathbb{R} \text{list}$$

$$sort \ [] = []$$

$$sort \ (h :: tl) = insert \ h \ (sort \ tl)$$
Sensitivity → Privacy
+ PL for Sensitivity = PL for Privacy

But wait, there’s more...
Probability distributions

For each type \( \tau \), let \( \bigcirc_\tau \) be the type of probability distributions over \( \tau \), with

\[
d_{\bigcirc_\tau}(\delta_1, \delta_2) = \frac{1}{\epsilon} \left( \max_{x \in \tau} \left| \ln \left( \frac{\delta_1(x)}{\delta_2(x)} \right) \right| \right)
\]

\[\text{add\_noise} : \mathbb{R} \rightarrow \bigcirc \mathbb{R}\]

[The definition measures how multiplicatively far apart two distributions are in the worst case, as is required by differential privacy.]
Typing relation

Typing contexts: \[ \Gamma ::= \cdot | \Gamma, x : \tau \text{ for } \tau \in \mathbb{R}_{>0} \cup \{\infty\} \]

"\(e\) is a well-formed expression of type \(\tau\) in a context \(\Gamma\):

\[ \Gamma \vdash e : \tau \]

A \(c\)-sensitive function of \(x\):

\[ x : c \tau_1 \vdash e : \tau_2 \]

More generally:

\[ x_1 : r_1 \tau_1, \ldots, x_n : r_n \tau_n \vdash e : \tau \]

[To have a hypothesis \(x : r \tau\) while constructing an expression \(e\) is to have permission to be \(r\)-sensitive to variation in the input \(x\): the output of \(e\) is allowed to vary by \(rs\) if the value substituted for \(x\) varies by \(s\).]

[in the second case the guarantee is that, if each \(x_i\) varies by \(s_i\), then the result of evaluating \(e\) only varies by \(\sum_i r_i s_i\). More carefully...]
Metric preservation

**Theorem** Suppose $\Gamma \vdash e : \tau$. Let sequences of values $(v_i)_{1 \leq i \leq n}$ and $(v'_i)_{1 \leq i \leq n}$ be given. Suppose for all $i \in 1, \ldots, n$ that we have

1. $\vdash v_i, v'_i : \tau_i$
2. $d_{\tau_i}(v_i, v'_i) = s_i$
3. $x_i : \tau_i \in \Gamma$.

If the program $[v_1/x_1] \cdots [v_n/x_n]e$ evaluates to $v$, then there exists a $v'$ such that $[v'_1/x_1] \cdots [v'_n/x_n]e$ evaluates to $v'$, and

$$d_{\tau}(v, v') \leq \sum_i r_i s_i.$$
DFuzz

Gaboardi, Haeberlen, Hsu, Narayan, and Pierce, Linear Dependent Types for Differential Privacy, POPL 2013

Talk to the authors this week!
Motivation

• Primitives of Fuzz are similar to PINQ in expressiveness
  • But many useful programs are not understood by the typechecker
• Main shortcoming: Cannot track data-dependent function sensitivity

\[ 2\text{iter}-k\text{-means} : !_{\infty} \mathbb{L}(\mathbb{R}^2) \to !_{f_{\infty}} \mathbb{R}^2 \text{ set} \to \mathcal{O}(\mathbb{L}(\mathbb{R}^2)) \]

**FIX:** L should be list
Plan

- Enrich type system of Fuzz with indexed types capable of tracking such data dependencies

\[ k\text{-means}: \forall i. (\text{vector of } \mathbb{N}[i] \to \mathbb{L}(\mathbb{R}^2)[k] \to \text{set of } \mathbb{R}^2 \to \text{set of } \mathbb{L}(\mathbb{R}^2)[k]) \]

- Ongoing work
  - Paper to appear in POPL 2013
  - Prototype implementation underway
  - Main challenge: Constraint solving

TODO: Figure out what the k-means type means!!
function kmeans
  (iters : Nat[i]) (eps : num[e])
  (db : [3 * i * e] (num, num) bag)
  (centers : list(num, num)[j])
  (iterate : num[e] -> (num, num) bag -o [3*e]
    list(num, num)[j] -> Circle list(num, num)[j])
  : Circle list(num, num)[j] {
    case iters of
      0     => return centers
    | n + 1 => sample next_centers =
      kmeans n eps db centers iterate;
      iterate eps db next_centers
  }
Status

- Prototype implementation underway
- Main issue: constraint solving
  - (hopefully using an SMT solver such as Z3)
Machine-Checked Proofs
Limitations (of language-based approaches)

- Each of the above approaches offers a fairly limited set of primitive datatypes (lists, bags, ...) and differentially private operations over them
  - The “reasons why” an algorithm is DP must be fairly straightforward

- Meanwhile, the algorithms community is continually generating clever new DP algorithms (often over other forms of data, e.g. graphs, streams)
Possible approaches

- Add new primitives
- Drop the demand that privacy proofs be generated automatically
  - this leads to...
CertiPriv

Barthe, Köpf, Olmedo, Béguelin, Probabilistic Relational Reasoning for Differential Privacy, POPL 2012
CertiPriv

- Allows reasoning about approximate quantitative properties of randomized computations
- Built from first principles and fully formalized in Coq
- Machine-checked proofs of differential privacy
  - Correctness of Laplacian and Exponential mechanisms
- State-of-art graph and streaming algorithms
DP for Interactive Systems


(Thanks to Anupam Datta for slides!)
Differential privacy is a definition for the functions used in the sanitizer.
System Model

- Bounded Memory
  - Cannot represent real numbers
  - Need discrete versions of privacy mechanisms
- Interactive I/O with environment
  - Answers queries over time
  - Also receiving new data points
- Probabilistic
Interaction Model

• Interleaving of data points, queries, and responses
• Mutable set of data points
• Adversary sees interleaving of queries and responses
• Differential noninterference generalizes both classical DP and classical noninterference from information-flow security
• Related to Pan Privacy [Dwork, Naor, Pitassi, Rothblum]
  • Maintains privacy for interactive systems under continual observation, even when the system’s internal state is observed
Proof Technique

• Use local properties to imply the global property of differential privacy
• Use a refinement lemma to relate abstract models to concrete implementations
• Decompose verification into two problems:
  • Prove that sanitization functions have differential privacy using absorbing Markov chains
  • Prove that system correctly store data points and use sanitization functions using *unwinding*
• Partially automated
Some other work in PL
Continuity of Programs

- Observes that many everyday programs are
  - continuous (i.e., arbitrarily small changes to their inputs only cause arbitrarily small changes to their outputs)
  - or Lipschitz continuous (i.e., when their inputs change, their outputs change at most proportionally).
- Proposes a mostly-automatic framework for verifying that a program is continuous or Lipschitz

Chaudhuri, Gulwani, and Lublinerman. Continuity analysis of programs. POPL 2010
DP in Process Calculi

- Consider a probabilistic process calculus as a specification formalism for concurrent systems
- Framework for reasoning about differential privacy in this setting
- Illustrate ideas on an anonymity-preservation property for (an extension of) the Crowds protocol

Xu, Modular Reasoning about Differential Privacy in a Probabilistic Process Calculus, manuscript 2012
Let me turn now to some related work in the systems area, where by “systems” I mean work where the main focus is on building something that actually works in practice. Not surprisingly, there is a lot to say in this domain, some of it strongly overlapping with the PL topics I’ve already mentioned (which makes sense, since programming languages also need to be implemented), some not. I can’t talk about everything that’s going on, but let me sketch a few pieces of work some of whose authors are here this week. (Sorry if I’ve missed some!)
Covert Channels


(Thanks to Andreas Haeberlen for slides!)
Covert-channel attacks

```java
noisy sum, foreach r in db, of {
    if (r.name=="Bob" && r.hasRating("Porn"))
        then {
            b=1;
        }
    return b
}
```

- The above query...
  - ... is differentially private (sensitivity zero!)
  - ... takes 1 second longer if the database contains Bob’s data
  - Result: Adversary learns private information with certainty!

- Other channels that can be exploited:
  - Global state
  - Privacy budget (!)
The attacks work in practice

- Both PINQ and Airavat are vulnerable

- What went wrong?
  - The authors were aware of this attack vector
  - Both papers discuss some ideas for possible defenses
  - But neither system has a defense that is fully effective
Threat model

- Too many channels!! Is it hopeless?
- Reasonable assumption: Querier is remote
- This leaves just two channels:
  - The actual answer to the query
  - The time until the answer arrives
Approach

- Close the remaining channels completely through a combination of systems and PL techniques

- **Language design** rules out state attacks etc.
  - Example: Simply don’t allow global variables!

- **Special runtime** to close the timing channel
Plugging the timing channel

- How to avoid leaking information via query completion time?
  - Could treat time as an additional output
  - **But:** Unclear how to determine sensitivity

- **Approach:** Make timing predictable
  - If time does not depend on the contents of the database, it cannot leak information
Timeouts and default values

- Querier specifies for each “microquery”:
  - a timeout $T$, and
  - a default value $d$

- Each time the microquery processes a row:
  - If completed in less than $T$, wait
  - If not yet complete at $T$, abort and proceed to next row
Predictable transactions

- **Isolation:** Microquery must not interfere with the rest of the computation in any way
  - E.g. by triggering garbage collector, changing runtime state, ...

- **Preemptability:** Must be able to abort microqueries at any time
  - Even in the middle of memory allocation, ...

- **Bounded deallocation:** Must be able to free any allocated resources within bounded time
  - Example: Microquery allocates lots of memory, acquires locks...
• talk about going to epsilon-delta DP
Dangers of Floating Point

float ≠ ℝ

• Duh...

Ilya Mironov, On Significance of the Least Significant Bits For Differential Privacy, CCS 2012
Distributed DP

Narayan and Haeberlen, \textit{Differentially private join queries over distributed databases}, OSDI 2012

(Thanks to Andreas for slide!)

See talk this week!
DJoin

- A differentially private query processor for distributed databases
- First practical solution that supports joins (with some restrictions)
- Based on two novel primitives
  - BN-PSI-CA: Differentially private set intersection cardinality
  - DCR: Denoise-combine-renoise
- Not fast enough for interactive use, but may be sufficient for offline data analysis
GUPT: Privacy Preserving Data Analysis Made Easy

Mohan, Thakurta, Shi, Song, and Culler. GUPT: privacy preserving data analysis made easy. SIGMOD 2012

Talk to authors this week!
**GUPT:** platform for differentially private execution of unmodified user code

1. **Improve output accuracy:** resampling, optimal block size estimation

2. **Usability:** describing privacy budget in terms of accuracy, privacy budget allocation

3. **Protection against side-channel attacks:** state attack, privacy-budget attack, timing attack

Also: a new model of data sensitivity that degrades privacy of data over time. Enables efficient allocation of different levels of privacy for different applications while guaranteeing an overall constant level of privacy and maximizing utility.
Main idea: Sample and Aggregate [NRS07, Smith11]

Data Set

D_1 f D_2 f D_3 f D_4 f \ldots D_k f

Isolated Execution Chambers
Winding Up...
Challenges

• Balancing expressiveness and automation
• Bullet-proof implementations
• Extending the tools with a broader range of data structures (graphs, streams) and DP algorithms

• Realistic examples!
Thank you!

Any questions?

http://privacy.cis.upenn.edu