Formal Verification of Differentially Private Mechanisms

Marco Gaboardi
University at Buffalo, SUNY
Goal of formal verification: building programs that are correct.
Why correctness matters?
Why correctness matters?

An example:
DARPA HACMS (High Assurance Cyber Military Systems)
What does “correct” mean?

In traditional program verification, a program is correct if it respects the specification:

• What is computed (functional aspects)
• How it is computed (non-functional aspects).

What does correct mean for differentially private applications?
A vast amount of individuals' data is collected, stored and accessed every day: demographic data, hospital records, location data, and much more. These data are valuable for scientific and medical research, for decision making, etc. However, access to data carries concerns for the privacy of the individuals contributing their data. These concerns restrict the way this vast amount of information can be used, and released.

Privacy, Accuracy, Efficiency

Figure 1: Data analysis triangle: privacy is the base, but accuracy and efficiency are critical.

The research on privacy-preserving technology is very active and several approaches have been proposed. An approach that emerged early in the statistical literature is based on the idea of using randomness to protect sensitive information [66]. By using randomness one can guarantee a protection to individual's data in the form of uncertainty—an attacker aiming to obtain some individual's data can be confused by this uncertainty. However, using randomness also changes the accuracy of the information extracted from the data, potentially compromising the validity of conclusions drawn from the data analysis.

More generally, the Fundamental law of Information Reconstruction [26, 30] informally says that if an adversary can observe the results of too many, too accurate statistics, then she can reconstruct with high probability the entire data. This law gives a mathematical meaning to the tension between the privacy of individuals and the accuracy of statistics. Implicit in the notion of accuracy of a data analysis is the number of data samples used: more samples typically means more accuracy, whereas fewer samples may lead to more efficient analysis (in terms of acquiring or using data). The number of samples is also related to privacy: more samples make it easier to protect the privacy of an individual sample.

We will focus on the three aspects of privacy, accuracy, and efficiency of data analyses. We see these three aspects as edges of a “data analysis triangle” (Figure 1). Our main focus is privacy—the base of this triangle—but the triangle (the data analysis) requires the other two edges: accuracy and efficiency. When considering data analysis, we believe it is fundamental to consider all three of these aspects.

Designing, implementing, and reasoning about data analyses is difficult. Modern data analyses are often based on subtle use of sophisticated algorithmic ideas using randomization, and concepts from probability and learning theory. Reasoning about their properties can be tedious and error-prone. Moreover, due to the quantitative and noisy nature of data analyses, it can be difficult to use implementations of a data analysis to discover flaws in the analysis. For these reasons, several formal verification techniques have been developed to help design, implement, and reason about data analyses.

Type system based verification approaches are particularly useful as they permit modular reasoning about program components. Approaches based on type systems have helped verifying the privacy of several mechanisms, but many basic mechanisms still escape this technique. Moreover, type system approaches (and other verification techniques) have so far focused on privacy, mostly neglecting accuracy and efficiency. There is thus a gap between how researchers and practitioners reason manually about their analyses and how formal verification can support this reasoning.

In order to fill this gap, we will develop foundational formal verification techniques to reason in a combined way about privacy, accuracy, and efficiency. We will do so by extending type systems techniques to reason about accuracy and efficiency in addition to privacy. We will also strengthen the support that these techniques offer for privacy. Our long term goal is to provide a suite of verification techniques that can be used by researchers and practitioners to guarantee private, accurate, and efficient analyses.

Broader Impacts:

The proposed research will develop foundational methods for privacy-preserving technology. Additionally, the project will develop educational material and tools that will help students, researchers, and practitioners approach the different aspects of data analysis in a combined way. The project will support one graduate student.
Abstract? or Concrete?
Desiderata: building private, accurate, and efficient implementations that are secure and resilient to attacks.
Byproduct

Systems that can help with the design of differentially private data analysis.
Outline

- Few words on program verification,
- Challenges in the verification of differential privacy,
- Verification methods developed so far,
- Looking forward.
A 10 thousand ft view on program verification…
Proofs vs Formal Proofs

Verification Tool

Proof

P

yes?

no?
Verification tools

- + expert provided annotations

Verification tools

(semi)-decision procedures
(SMT solvers, ITP)
An example

Consider a simple program squaring a given number $m$:

```
{ X = m }
Y := 0 ; ;
Z := 0 ; ;
WHILE Y ≠ X DO
    Z := Z + X ; ;
    Y := Y + 1
END
{ Z = m*m }
```
An example

A proof of correctness can be given as follows:

\[
\begin{align*}
\{ & X = m \} \implies \\
& \{ 0 = 0 \cdot m \land X = m \} \\
Y & := 0; \\
& \{ 0 = Y \cdot m \land X = m \} \\
Z & := 0; \\
& \{ Z = Y \cdot m \land X = m \} \\
\text{WHILE} & \ Y \neq X \ \text{DO} \\
& \{ Z = Y \cdot m \land X = m \land Y \neq X \} \implies \\
& \{ Z + X = (Y + 1) \cdot m \land X = m \} \\
Z & := Z + X; \\
& \{ Z = (Y + 1) \cdot m \land X = m \} \\
Y & := Y + 1 \\
& \{ Z = Y \cdot m \land X = m \} \\
\text{END} \\
& \{ Z = Y \cdot m \land X = m \land \neg (Y \neq X) \} \implies \\
& \{ Z = m \cdot m \}
\end{align*}
\]
Questions that program verification can help with

- Are our algorithms bug-free?
- Do implementations respect the algorithms?
- Is the system architecture bug-free?
- Is the code efficient?
- Is the actual machine code correct?
- Do the optimization preserve correctness?
- Is the full stack attack-resistant?
Some successful stories - 1

- CompCert - a fully verified C compiler,
- Sel4, CertiKOS - formal verification of OS kernel
- A formal proof of the Odd order theorem,
- A formal proof of Kepler conjecture.

Years of work from very specialized researchers!
Some successful stories - II

- Automated verification for Integrated Circuit Design.
- Automated verification for Floating point computations,
- Automated verification of Boeing flight control - Astree,
- Automated verification of Facebook code - Infer.

The years of work go in the design of the techniques!
Verification trade-offs

- required expertise
- expressivity
- granularity of the analysis
How things can go wrong in Differential Privacy....
The challenges of differential privacy

Given $\varepsilon, \delta \geq 0$, a mechanism $M: db \rightarrow O$ is $(\varepsilon, \delta)$-differentially private iff

\[ \forall b_1, b_2 : db \text{ differing in one record and } \forall S \subseteq O: \]
\[ \Pr[M(b_1) \in S] \leq \exp(\varepsilon) \cdot \Pr[M(b_2) \in S] + \delta \]

• Relational reasoning,
• Probabilistic reasoning,
• Quantitative reasoning
Example 1: the sparse vector case

<table>
<thead>
<tr>
<th>Algorithm 1 An instantiation of the SVT proposed in this paper.</th>
<th>Algorithm 2 SVT in Dwork and Roth 2014 [8].</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $D, Q, \Delta, T = T_1, T_2, \ldots , c$.</td>
<td><strong>Input:</strong> $D, Q, \Delta, T, c$.</td>
</tr>
<tr>
<td>1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap}(\Delta/\epsilon_1)$</td>
<td>1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap}(\Delta/\epsilon_1)$</td>
</tr>
<tr>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0</td>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0</td>
</tr>
<tr>
<td>3: for each query $q_i \in Q$ do</td>
<td>3: for each query $q_i \in Q$ do</td>
</tr>
<tr>
<td>4: $\nu_i = \text{Lap}(2c\Delta/\epsilon_2)$</td>
<td>4: $\nu_i = \text{Lap}(2c\Delta/\epsilon_2)$</td>
</tr>
<tr>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
</tr>
<tr>
<td>6: Output $a_i = T$</td>
<td>6: Output $a_i = T$, $\rho = \text{Lap}(\epsilon\Delta/\epsilon_2)$</td>
</tr>
<tr>
<td>7: count = count + 1, <strong>Abort</strong> if count $\geq c.$</td>
<td>7: count = count + 1, <strong>Abort</strong> if count $\geq c.$</td>
</tr>
<tr>
<td>8: else</td>
<td>8: else</td>
</tr>
<tr>
<td>9: Output $a_i = \bot$</td>
<td>9: Output $a_i = \bot$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 3 SVT in Roth’s 2011 Lecture Notes [15].</th>
<th>Algorithm 4 SVT in Lee and Clifton 2014 [13].</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $D, Q, \Delta, T, c$.</td>
<td><strong>Input:</strong> $D, Q, \Delta, T, c$.</td>
</tr>
<tr>
<td>1: $(\epsilon_1 = \epsilon/2, \rho = \text{Lap}(\Delta/\epsilon_1),$</td>
<td>1: $(\epsilon_1 = \epsilon/4, \rho = \text{Lap}(\Delta/\epsilon_1)$</td>
</tr>
<tr>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0</td>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0</td>
</tr>
<tr>
<td>3: for each query $q_i \in Q$ do</td>
<td>3: for each query $q_i \in Q$ do</td>
</tr>
<tr>
<td>4: $\nu_i = \text{Lap}(\epsilon\Delta/\epsilon_2)$</td>
<td>4: $\nu_i = \text{Lap}(\Delta/\epsilon_2)$</td>
</tr>
<tr>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
</tr>
<tr>
<td>6: Output $a_i = q_i(D) + \nu_i$</td>
<td>6: Output $a_i = T$</td>
</tr>
<tr>
<td>7: count = count + 1, <strong>Abort</strong> if count $\geq c.$</td>
<td>7: count = count + 1, <strong>Abort</strong> if count $\geq c.$</td>
</tr>
<tr>
<td>8: else</td>
<td>8: else</td>
</tr>
<tr>
<td>9: Output $a_i = \bot$</td>
<td>9: Output $a_i = \bot$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 5 SVT in Stoddard et al. 2014 [18].</th>
<th>Algorithm 6 SVT in Chen et al. 2015 [1].</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $D, Q, \Delta, T$.</td>
<td><strong>Input:</strong> $D, Q, \Delta, T = T_1, T_2, \ldots$.</td>
</tr>
<tr>
<td>1: $(\epsilon_1 = \epsilon/2, \rho = \text{Lap}(\Delta/\epsilon_1)$</td>
<td>1: $(\epsilon_1 = \epsilon/2, \rho = \text{Lap}(\Delta/\epsilon_1)$</td>
</tr>
<tr>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$</td>
<td>2: $\epsilon_2 = \epsilon - \epsilon_1$</td>
</tr>
<tr>
<td>3: for each query $q_i \in Q$ do</td>
<td>3: for each query $q_i \in Q$ do</td>
</tr>
<tr>
<td>4: $\nu_i = 0$</td>
<td>4: $\nu_i = \text{Lap}(\Delta/\epsilon_2)$</td>
</tr>
<tr>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
<td>5: if $q_i(D) + \nu_i \geq T_i + \rho$ then</td>
</tr>
<tr>
<td>6: Output $a_i = T$</td>
<td>6: Output $a_i = T$</td>
</tr>
<tr>
<td>7:</td>
<td>7:</td>
</tr>
<tr>
<td>8: else</td>
<td>8: else</td>
</tr>
<tr>
<td>9: Output $a_i = \bot$</td>
<td>9: Output $a_i = \bot$</td>
</tr>
</tbody>
</table>
Example 2: the rounding case

- Attack based on irregularities of floating point implementations of the Laplace mechanism,
- A solution: snapping mechanism
- How about other mechanisms?

Ilya Mironov:
On significance of the least significant bits for differential privacy. ACM CCS 2012
Example 3: the floating point case

- Timing attack based on x86 difference of addition/multiplication running time difference,

- A solution: a constant time library.

What we have so far…
A 10 thousand ft view on program verification

- Verification tools

+ Expert provided annotations

- (semi)-decision procedures (SMT solvers, ITP)
Verification tools

• They handle well logical formulas, numerical formulas and their combination,

• They offer limited support for probabilistic reasoning.

We need a good abstraction of the problem.
Compositional Reasoning about the Privacy Budget

Sequential Composition
Let $M_i$ be $\epsilon_i$-differentially private ($1 \leq i \leq k$).
Then $M(x) = (M_1(x), \ldots, M_k(x))$ is $\sum_{i=0}^{k} \epsilon_i$.

- We can reason about the privacy budget,
- If we have basic components for privacy we can just focus on counting,
- It requires a limited reasoning about probabilities,
- Implemented in different tools, e.g. PINQ (McSherry’10), Airavat (Roy’10), etc.
Compositional reasoning about sensitivity

\[ GS(f) = \max_{v \sim v'} |f(v) - f(v')| \]

- It allows to decompose the analysis/construction of a DP program,
- It requires a limited reasoning about probabilities,
- Similar reasoning as basic composition.
- Implemented using type-checking in Fuzz (Reed&Pierce’10),
- Recently extended to AdaptiveFuzz (Winograd-cort&co’17).
Reasoning about DP via Approximate Probabilistic

- Generalize pointwise-observations to other relations allowing more general relational reasoning,
- More involved reasoning about divergences,
- Formal proof of the correctness of sparse vector,
- Implemented in EasyCrypt and HOARRe² (Barthe et al.’13, ’15)
- Recently extended to zCDP, RDP (Sato et al.’17)
- New, fully automated version (Albarghouthi & Hsu’17)
Semi-automated DP proofs using Randomness Assignments

- Permits to build more flexible reasoning about correspondences between the programs, and the privacy budget,
- requires few annotations and can be combined with other tools making it almost automated,
- the proof of sparse vector only requires 2 lines of annotations,
- implemented in LightDP (Zhang & Kifer’17)

injective map producing the same output
Other works

- Bisimulation based methods (Tschantz&al - Xu&al)
- Fuzz with distributed code (Eigner&Maffei)
- Satisfiability modulo counting (Friedrikson&Jha)
- Bayesian Inference (BFGGHS)
- Accuracy bounds (BGGHS)
- Continuous models (Sato)
- zCDP (BGHS)
- ....
- Many other systems.
Looking forward...
Abstract? or Concrete?
Basic Mechanism Implementation

- We aim at verifying end-to-end a basic, realistic mechanism (from the algorithm to the code),
- We focus on a mechanism for the local model of differential privacy (simpler mechanisms, practically relevant),
- We are looking at mechanisms that have good privacy-utility tradeoff, and are efficient,
- We focus first on a machine independent approach, and add consider more concrete models later.
Private Heavy Hitter

- We focus on algorithms for the heavy hitter problem: practically relevant and a availability of several different algorithms,
- We are implementing the TreeHist algorithm by Bassily et al.’17 which provides a good accuracy and is efficient.
- The privacy guarantee is obtained through a simple randomized response mechanism,
- It makes non trivial transformations both on the client and server side.
Our approach

- Formal Logic based on coupling
- Foundational Cryptography Framework
  - Petcher&Morrisett'15

- Coq proof assistant

- Verifiable C language & program logic
  - VST retargetable Separation Logic

- COMPCERT verified C compiler (from INRIA)

- Recently used for HMAC for OpenSSL, (part of) TLS.
Expected Outcomes

• Many months of work!
• Increasing the confidence on the correctness of the mechanism implementation,
• Development of techniques for proving correct basic mechanisms from the local model.
Thanks