Differential Privacy and Verification

Marco Gaboardi
University at Buffalo, SUNY
Given a program $P$, is it differentially private?
P

Verification Tool

yes?

no?
Given a differentially private program $P$, does it maintain its accuracy promises?
Given a differentially private program $P$ that maintains its accuracy promises, can we guarantee that it is also efficient?
Algorithm 2 DualQuery

**Input:** Database $D \in \mathbb{R}^{||X||}$ (normalized) and linear queries $q_1, \ldots, q_k \in \{0, 1\}^{||X||}$. 

**Initialize:** Let $Q = \bigcup_{j=1}^k q_j \cup \overline{q_j}$, $Q^1$ uniform distribution on $Q$,

$$T = \frac{16 \log |Q|}{\alpha^2}, \quad \eta = \frac{\alpha}{4}, \quad s = \frac{48 \log \left( \frac{2|X|T}{\beta} \right)}{\alpha^2}.$$

For $t = 1, \ldots, T$:
- Sample $s$ queries $\{q_i\}$ from $Q$ according to $Q^t$.
- Let $\overline{q} := \frac{1}{s} \sum_i q_i$.
- Find $x^t$ with $\langle \overline{q}, x^t \rangle \geq \max_x \langle \overline{q}, x \rangle - \alpha/4$.

**Update:** For each $q \in Q$:
- $Q^{t+1}_q := \exp(-\eta \langle q, x^t - D \rangle) \cdot Q^t_q$.
- Normalize $Q^{t+1}$.

Output synthetic database $\hat{D} := \bigcup_{t=1}^T x^t$.  

---

**4.1 Privacy**

The privacy proofs are largely routine, based on the composition theorems. Rather than fixing "and solving for the other parameters, we present the privacy cost as function of parameters $T, s, \eta$.

Later, we will tune these parameters for our experimental evaluation. We will use the privacy of the following mechanism (due to McSherry and Talwar [26]) as an ingredient in our privacy proof.

**Definition 4.1** (McSherry and Talwar [26]).

Given some arbitrary output range $R$, the exponential mechanism selects and outputs an element $r \in R$ with probability proportional to $\exp(\alpha S(D, r))$, where $\alpha$ is the sensitivity of $S$, defined as $\alpha = \max_D, D_0 : |D - D_0| = 1$.

The exponential mechanism is "\(-differentially private."

We first prove pure "\(-differential privacy.

**Theorem 4.2.** DualQuery is "\(-differentially private for $\eta = \frac{T}{s}$."

**Proof.** We will argue that sampling from $Q^t$ is equivalent to running the exponential mechanism with some quality score. At round $t$, let $\{x_i\}$ for $i \in [t]$ be the best responses for the previous rounds. Let $r(q, D) = t \sum_{i=1}^t \langle q, x_i \rangle - \alpha/4$. 

For each $q \in Q$:
- $Q^{t+1}_q := \exp(-\eta \langle q, x^t - D \rangle) \cdot Q^t_q$.
- Normalize $Q^{t+1}$.

Output synthetic database $\hat{D} := \bigcup_{t=1}^T x^t$. 

---

**An algorithm**
A program

https://github.com/ejgallego/dualquery/
Some issues

- Are the algorithms bug-free?
- Do the implementations respect their specifications?
- Is the system architecture bug-free?
- Is the code efficient?
- Do the optimization preserve privacy and accuracy?
- Is the actual machine code correct?
- Is the full stack attack-resistant?
Outline

• Few more words on program verification,
• Challenges in the verification of differential privacy,
• Few verification methods developed so far,
• Looking forward.
Carnegie Mellon mistakenly accepts 800 applicants, then rejects them

By Holly Yan and Katia Hetter, CNN

Updated 2:27 PM ET, Wed February 18, 2015
### Algorithm 1 An instantiation of the SVT proposed in this paper.

**Input:** $D, Q, \Delta, T = T_1, T_2, \ldots, c$.
1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap} (\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0
3: for each query $q_i \in Q$ do
4: $\nu_i = \text{Lap} (2c\Delta/\epsilon_2)$
5: if $q_i(D) + \nu_i \geq T_i + \rho$ then
6: Output $a_i = \top$
7: count = count + 1, **Abort** if count $\geq c$.
8: else
9: Output $a_i = \perp$

### Algorithm 2 SVT in Dwork and Roth 2014 [8].

**Input:** $D, Q, \Delta, T, c$.
1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap} (c\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0
3: for each query $q_i \in Q$ do
4: $\nu_i = \text{Lap} (2c\Delta/\epsilon_1)$
5: if $q_i(D) + \nu_i \geq T + \rho$ then
6: Output $a_i = \top$, $\rho = \text{Lap} (c\Delta/\epsilon_2)$
7: count = count + 1, **Abort** if count $\geq c$.
8: else
9: Output $a_i = \perp$

### Algorithm 3 SVT in Roth’s 2011 Lecture Notes [15].

**Input:** $D, Q, \Delta, T, c$.
1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap} (\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0
3: for each query $q_i \in Q$ do
4: $\nu_i = \text{Lap} (c\Delta/\epsilon_2)$
5: if $q_i(D) + \nu_i \geq T + \rho$ then
6: Output $a_i = q_i(D) + \nu_i$
7: count = count + 1, **Abort** if count $\geq c$.
8: else
9: Output $a_i = \perp$

### Algorithm 4 SVT in Lee and Clifton 2014 [13].

**Input:** $D, Q, \Delta, T, c$.
1: $\epsilon_1 = \epsilon/4$, $\rho = \text{Lap} (\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$, count = 0
3: for each query $q_i \in Q$ do
4: $\nu_i = \text{Lap} (\Delta/\epsilon_2)$
5: if $q_i(D) + \nu_i \geq T + \rho$ then
6: Output $a_i = \top$
7: count = count + 1, **Abort** if count $\geq c$.
8: else
9: Output $a_i = \perp$

### Algorithm 5 SVT in Stoddard et al. 2014 [18].

**Input:** $D, Q, \Delta, T$.
1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap} (\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$
3: for each query $q_i \in Q$ do
4: $\nu_i = 0$
5: if $q_i(D) + \nu_i \geq T + \rho$ then
6: Output $a_i = \top$
7: else
8: Output $a_i = \perp$

### Algorithm 6 SVT in Chen et al. 2015 [1].

**Input:** $D, Q, \Delta, T = T_1, T_2, \ldots$.
1: $\epsilon_1 = \epsilon/2$, $\rho = \text{Lap} (\Delta/\epsilon_1)$
2: $\epsilon_2 = \epsilon - \epsilon_1$
3: for each query $q_i \in Q$ do
4: $\nu_i = \text{Lap} (\Delta/\epsilon_2)$
5: if $q_i(D) + \nu_i \geq T_i + \rho$ then
6: Output $a_i = \top$
7: else
8: Output $a_i = \perp$

### Table: Algorithm Variants

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\epsilon_1$</th>
<th>$\epsilon/2$</th>
<th>$\epsilon/2$</th>
<th>$\epsilon/2$</th>
<th>$\epsilon/4$</th>
<th>$\epsilon/2$</th>
<th>$\epsilon/2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale of threshold noise $\rho$</td>
<td>$\Delta/\epsilon_1$</td>
<td>$c\Delta/\epsilon_1$</td>
<td>$\Delta/\epsilon_1$</td>
<td>$\Delta/\epsilon_1$</td>
<td>$\Delta/\epsilon_1$</td>
<td>$\Delta/\epsilon_1$</td>
<td></td>
</tr>
<tr>
<td>Reset $\rho$ after each output of $\top$ (unnecessary)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Scale of query noise $\nu_i$</td>
<td>2$c\Delta/\epsilon_2$</td>
<td>2$c\Delta/\epsilon_2$</td>
<td>2$c\Delta/\epsilon_2$</td>
<td>2$c\Delta/\epsilon_2$</td>
<td>0</td>
<td>2$c\Delta/\epsilon_2$</td>
<td></td>
</tr>
<tr>
<td>Outputting $q_i + \nu_i$ instead of $\top$ (not private)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Outputting unbounded $\top$’s (not private)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Privacy Property</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
<td>$\epsilon$-DP</td>
</tr>
</tbody>
</table>

### Notes

- **Alg. 1**: An instantiation of the SVT proposed in this paper.
- **Alg. 2**: SVT in Dwork and Roth 2014 [8].
- **Alg. 3**: SVT in Roth’s 2011 Lecture Notes [15].
- **Alg. 4**: SVT in Lee and Clifton 2014 [13].
- **Alg. 5**: SVT in Stoddard et al. 2014 [18].
- **Alg. 6**: SVT in Chen et al. 2015 [1].

- $\epsilon_1$: Threshold for sensitivity normalization.
- $\epsilon$: Privacy parameter.
- $\Delta$: Scale of threshold noise.
- $\rho$: Scale of query noise.
- $\nu_i$: Query noise for each query $q_i$.
- $\top$: Output 1.
- $\perp$: Output 0.
Some successful stories - I

- 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 (lead by T. Hales).
Some successful stories - I

- 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 (lead by T. Hales).

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.
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
What program verification isn’t…

- Algorithm design,
- Trial and error,
- Program testing,
- System engineering,
- A certification process.
What program verification can help with…

- Designing languages for non-experts,
- Guaranteeing the correctness of algorithms,
- Guaranteeing the correctness of code,
- Designing automated techniques for guaranteeing differential privacy,
- Help providing tools for certification process.
The challenges of differential privacy

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

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

- Relational reasoning,
- Probabilistic reasoning,
- Quantitative reasoning
A 10 thousand ft view on program verification

- Verification tools
- Expert provided annotations
- (semi)-decision procedures (SMT solvers, ITP)
Work-flow

- mechanism described as a program
  - expert manually adds assertions
- program annotated with assertions
  - proof checker generates VCs
- whole set of VCs
  - automatic solver checks VCs
  - VCs not solvable automatically
    - interactive solver checks VCs
- verified mechanism

VCs = Verification Conditions
Semi-decision procedures

• Require a good decomposition of the problem,

• Handle well logical formulas, numerical formulas and their combination,

• Limited support for probabilistic reasoning (usually through decision procedures for counting).
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 DP programs by monitoring the privacy budget,

• If we have basic components for privacy we can just focus on counting,

• It requires a limited reasoning about probabilities,

• This way of reasoning adapt to other compositions.
Iterated - CDF

CDF(X) = number of records with value ≤ X.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Code</th>
<th>Disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>29</td>
<td>19144</td>
<td>diabets</td>
</tr>
<tr>
<td>Bob</td>
<td>48</td>
<td>19146</td>
<td>tumor</td>
</tr>
<tr>
<td>Jim</td>
<td>25</td>
<td>34505</td>
<td>flue</td>
</tr>
<tr>
<td>Alice</td>
<td>62</td>
<td>19144</td>
<td>diabets</td>
</tr>
<tr>
<td>Bill</td>
<td>39</td>
<td>16544</td>
<td>anemia</td>
</tr>
<tr>
<td>Sam</td>
<td>61</td>
<td>19144</td>
<td>diabets</td>
</tr>
</tbody>
</table>

CDF(Xn) List of buckets

CDF(50) = 4
CDF(40) = 3
CDF(30) = 2
PINQ-like languages - McSherry

it-CDF (raw : data) (budget : R) (buckets : list) (ε : R)
  : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
  {
    b = db.where(y => y.val ≤ b).noisyCount(ε);
    yield return b;
  }
}
it-CDF (raw : data) (budget : R) (buckets : list) (ε : R) : list
{
    var agent = new PINQAgentBudget(budget);
    var db = new PINQueryable<data>(rawdata, agent);
    foreach (var b in buckets)
    {
        b = db.where(y => y.val ≤ b).noisyCount(ε);
        yield return b;
    }
}
PINQ-like languages - McSherry

```csharp
it-CDF (raw : data) (budget : R) (buckets : list) (ε : R)
  : list
{
  var agent = new PINQAgentBudget(budget);
  var db = new PINQueryable<data>(rawdata, agent);
  foreach (var b in buckets)
    b = db.where(y => y.val ≤ b).noisyCount(ε);
  yield return b;
}
```

raw data are accessed through a PINQueryable
we have transformations (scaling factor)
PINQ-like languages - McSherry

```csharp
it-CDF (raw : data) (budget : R) (buckets : list) (\(\varepsilon\) : R) : list
{
    var agent = new PINQAgentBudget(budget);
    var db = new PINQueryable<data>(rawdata, agent);
    foreach (var b in buckets)
    {
        b = db.where(y => y.val \leq b).noisyCount(\(\varepsilon\));
        yield return b;
    }
}
```

we have transformations (scaling factor)

aggregate operations (actual budget consumption)
Enough budget?

\[
\text{it-CDF} \ (\text{raw : data}) \ (\text{budget : R}) \ (\text{buckets : list}) \ (\varepsilon : R) \\
\text{ : list}
\]

{ 
    var agent = new PINQAgentBudget(budget);
    var db = new PINQueryable<data>(rawdata, agent);
    foreach (var b in buckets)
    
        b = db.where(y => y.val \leq b).noisyCount(\varepsilon);
    yield return b;
}
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,
- A metric property of the function (DMNS06),
- It requires a limited reasoning about probabilities,
- Similar worst case reasoning as basic composition.
Fuzz-like languages - Penn

\[
\text{it-CDF} \ (b : \text{data}) \ (\text{buckets} : \text{list}) : \text{list} \\
\{ \\
\quad \text{case buckets of} \\
\quad \quad |\text{nil} \Rightarrow \text{nil} \\
\quad \quad |x::xs \Rightarrow \text{size} \ (\text{filter} \ (\text{fun} \ y \Rightarrow y \leq x) \ b)) \\
\quad \quad \quad :: \ \text{it-CDF} \ xs \ b \\
\} 
\]
Fuzz-like languages - Penn

```
let it-CDF (b : [??] data) (buckets : list) : list =
  let rec aux buckets =
    match buckets with
    | [] => nil
    | x :: xs -> size (filter (fun y => y <= x) b)) :: aux xs
  in aux buckets
```

How sensitive?
Fuzz-like languages - Penn

\[ \text{let-CDF} \ (b : \text{data}) \ (\text{buckets} : \text{list}) : \text{list} \ \\
\{ \ \\
\quad \text{case buckets of} \ \\
\qquad | \text{nil} \ => \text{nil} \ \\
\qquad | x::xs => \text{size} (\text{filter} (\text{fun} \ y => y \leq x) \ b)) \ \\
\quad \quad :: \text{let-CDF} \ xs \ b \ \\
\} \ \\
\]

Let's assume \( |- \text{size} : [1]\text{data} \rightarrow \text{O} \ \text{R} \)
Similarly, \(-\ filter : [\infty]prop \rightarrow [1]data \rightarrow R\)
Fuzz-like languages - Penn

```
\text{it-CDF} \ (b : [??] \ data) \ (\text{buckets} : \text{list}) : \text{list} \\
\{ \\
\text{case buckets of} \\
|\text{nil} \ => \text{nil} \\
|x::xs \ => \text{size} \ (\text{filter} \ (\text{fun} \ y \Rightarrow y \leq x) \ b)) \\
\} :: \text{it-CDF} \ xs \ b \\
(b : [0] \ data) \\
(b : [n+1] \ data)
```
it-CDF (b : [n] data) (buckets : list[n]) : list
{
    case buckets of
    | nil    => nil
    | x::xs  => size (filter (fun y => y ≤ x) b))
        :: it-CDF xs b
}

n-sensitive!
Fuzz-like languages - Penn

\[
\text{it-CDF} (b : [\varepsilon^* n] \text{ data}) (\text{buckets : list}[n]) (\varepsilon: \text{num}): \text{nlist}
\{
\text{case buckets of}
\begin{align*}
|\text{nil} & \Rightarrow \text{nil} \\
|\text{x::xs} & \Rightarrow \text{Lap } \varepsilon \text{ size (filter (fun y \Rightarrow y \leq x) b))}
\end{align*}
:: \text{it-CDF} \text{ xs b}
\}
\]
Reasoning about DP via probabilistic coupling - BGGHS

For two (sub)-distributions $\mu_1, \mu_2 \in \text{Dist}(A)$ we have an approximate coupling $\mu_1 C_{\epsilon, \delta}(R) \mu_2$ iff there exists $\mu \in \text{Dist}(A \times A)$ s.t.

- $\text{supp}\mu \subseteq R$
- $\pi_i \mu \leq \mu_i$
- $\max_A (\pi_i \mu - e^\epsilon \mu_i, \mu_i - e^\epsilon \pi_i \mu) \leq \delta$

- Generalize indistinguishability to other relations allowing more general relational reasoning,

- More involved reasoning about probability distances and divergences,

- Preserving the ability to use semi-decision logical and numerical procedures.
pRHL-like languages

CDF example similar to the previous ones

\[ b \sim b' \Rightarrow (\text{itcdf } b \ l \ \epsilon) \ C_{\epsilon,0}(=) \ (\text{itcdf } b' \ l \ \epsilon) \]
pRHL-like languages

CDF example similar to the previous ones

\[ b \sim b' \Rightarrow (\text{itcdf } b \ l \ \epsilon) \ C_{\epsilon,0}(=) \ (\text{itcdf } b' \ l \ \epsilon) \]

Having two copies of the program allows to compare different parts of the same program.
Why this helps?

It allows to internalize better the properties of Laplace

\[
\left( \text{Lap} \left( \frac{1}{\epsilon} v_1 \right) \right) \mathcal{C}_{|k + v_1 - v_2|, \epsilon, 0}(x_1 + k = x_2) \left( \text{Lap} \left( \frac{1}{\epsilon} v_2 \right) \right)
\]

can be used to assert symbolically several facts about probabilities.
Why this helps?

\[(\text{Lap} (1/\epsilon) v_1) \cdot C_{|v_1-v_2|\epsilon,0}(x_1 = x_2) \cdot (\text{Lap} (1/\epsilon) v_2)\]

expresses

\[|\log \left( \frac{\Pr(\text{Lap} (1/\epsilon) v_1 = r)}{\Pr(\text{Lap} (1/\epsilon) v_2 = r)} \right) | \leq |v_1 - v_2|\epsilon\]
Why this helps?

\[ |v_1 - v_2| \leq k \Rightarrow (\text{Lap}(1/\epsilon) v_1) \mathcal{C}_{2k\epsilon,0}(x_1 + k = x_2) (\text{Lap}(1/\epsilon) v_2) \]

expresses

\[ |v_1 - v_2| \leq k \Rightarrow \left| \log \left( \frac{\Pr(\text{Lap}(1/\epsilon) v_1 = r + k)}{\Pr(\text{Lap}(1/\epsilon) v_2 = r)} \right) \right| \leq 2k\epsilon \]
Why this helps?

\[ (\text{Lap}(1/\epsilon) v_1) C_{0,0}(x_1 - x_2 = v_1 - v_2) (\text{Lap}(1/\epsilon) v_2) \]

expresses

\[ \left| \log \left( \frac{\Pr(\text{Lap}(1/\epsilon) v_2 + k = r + k)}{\Pr(\text{Lap}(1/\epsilon) v_2 = r)} \right) \right| \leq 0 \]
Other works

- Bisimulation based methods (Tschantz & al - Xu & al)
- Fuzz with distributed code (Eigner & Maffei)
- Satisfiability modulo counting (Friedrikson & Jha)
- Bayesian Inference (BFGGHS)
- Adaptive Fuzz (Penn)
- Accuracy bounds (BGGHS)
- Continuous models (Sato)
- Lightweight verification - injective function argument (Zhang & Kifer)
- Relational symbolic execution for R - generating DP counterexamples (Chong & Farina & Gaboardi)
- Formalizing the local model (Ebadi & Sands)
- zCDP (BGHS)
Challenges

- All of these tools are research projects and most of them are usable only by experts.

Can we use them to certify correct a library of basic mechanism?

Which non-expert we should aim for?
Other Challenges

- Are there other fundamental principles that we can use?
- How can we extend them to verify accuracy and efficiency?
- There are several works on the verification of randomness, floating points, SMC, etc. Can we combine the different approaches?
- How can we internalize more involved data models assumptions?
- From benchmarks to certification?