DP-Finder: Finding Differential Privacy Violations by Sampling and Optimization

Benjamin Bichsel
ETH Zurich, Switzerland
benjamin.bichsel@inf.ethz.ch

Timon Gehr
ETH Zurich, Switzerland
timon.gehr@inf.ethz.ch

Dana Drachsler-Cohen
ETH Zurich, Switzerland
dana.drachsler@inf.ethz.ch

Petar Tsankov
ETH Zurich, Switzerland
petar.tsankov@inf.ethz.ch

Martin Vechev
ETH Zurich, Switzerland
martin.vechev@inf.ethz.ch

ABSTRACT

We present `DP-Finder`, a novel approach and system that automatically derives lower bounds on the differential privacy enforced by algorithms. Lower bounds are practically useful as they can show tightness of existing upper bounds or even identify incorrect upper bounds. Computing a lower bound involves searching for a counterexample, defined by two neighboring inputs and a set of outputs, that identifies a large privacy violation. This is an inherently hard problem as finding such a counterexample involves inspecting a large (usually infinite) and sparse search space.

To address this challenge, `DP-Finder` relies on two key insights. First, we introduce an effective and precise correlated sampling method to estimate the privacy violation of a counterexample. Second, we show how to obtain a differentiable version of the problem, enabling us to phrase the search task as an optimization objective to be maximized with state-of-the-art numerical optimizers. This allows us to systematically search for large privacy violations.

Our experimental results indicate that `DP-Finder` is effective in computing differential privacy lower bounds for a number of randomized algorithms. For instance, it finds tight lower bounds in algorithms that obfuscate their input in a non-trivial fashion.

CCS CONCEPTS

• **Security and privacy**; • **Mathematics of computing** → *Probability and statistics*;

KEYWORDS

Differential privacy; Lower bounds; Sampling; Optimization

ACM Reference Format:
Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin Vechev. 2018. DP-Finder: Finding Differential Privacy Violations by Sampling and Optimization. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3243734.3243863

1 INTRODUCTION

Differential privacy (DP) [12] has emerged as an important property that measures the amount of leaked information by (randomized) algorithms [2, 7, 14, 21, 22, 24, 27, 29, 31]. Informally, a randomized algorithm is ϵ-differentially private (denoted ϵ-DP) if the distance between the output distributions it produces, for any two neighboring inputs, is bounded by ϵ. The standard way to enforce ϵ-DP is to add noise to computations, where the amount of noise is determined from the bound ϵ that the user would like to enforce. Determining the *exact* bound ϵ enforced by a randomized algorithm is important for two reasons. First, a conservative upper bound requires users to add more noise than necessary. Second, an incorrect bound (lower than the one actually enforced) may result in adding an insufficient amount of noise, thereby leading to actual privacy violations.

To address these issues, ideally one would derive exact privacy bounds enforced by randomized algorithms. Unfortunately, proving exact privacy bounds is a challenging task. Thus, in practice, existing approaches are limited to deriving upper bounds on the differential privacy enforced by algorithms [3, 4, 6, 15, 27, 32, 35]. These upper bounds are sometimes conservative, and have led to follow-up works that improved them (e.g., [17]). Moreover, in some cases, these upper bounds were incorrect, resulting in algorithms that *did not* enforce ϵ-DP. For example, [26] showed that a supposedly differentially private variants of the Sparse Vector Technique were actually not differentially private.

Finding Lower Bounds. In this work, we study the task of computing *lower bounds* on the ϵ-DP of randomized algorithms. That is, our goal is to find the largest ϵ for which a randomized algorithm is *not* ϵ-DP. Finding such lower bounds is practically useful and, combined with previous results on finding upper bounds, can be used to determine exactness or, alternatively, suggest that the upper bound can be improved. Moreover, discovering lower bounds provides an effective tool for testing the correctness of established upper bounds, by searching for lower bounds that may exceed them.

Challenges. The task of finding DP violations introduces two challenges. First, it requires efficiently estimating the violation induced by particular inputs and a set of outputs. This involves reasoning about probabilities which are difficult to compute analytically. Second, it requires finding inputs and a set of outputs that induce large privacy violations. This search involves solving a complex, non-differentiable maximization problem in a search space where few inputs and set of outputs induce large privacy

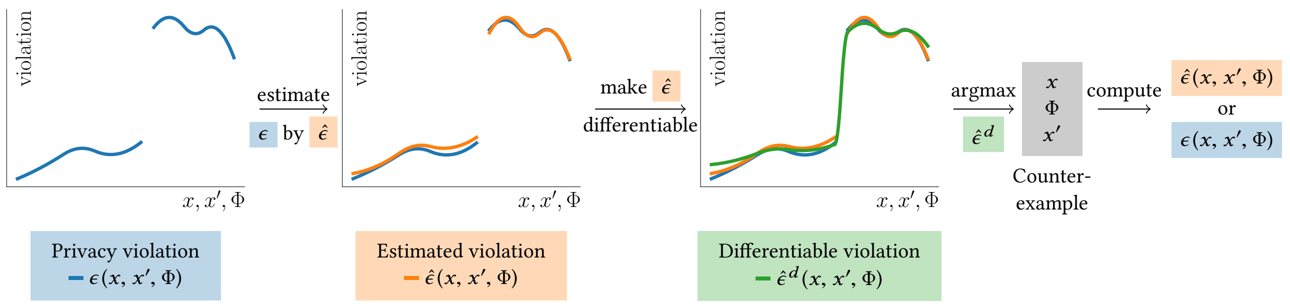


Figure 1: A conceptual overview of our approach. Our goal is to search for the maximal privacy violation $\epsilon(x, x', \Phi)$, a function which is hard to compute in general, and non-differentiable. `DP-Finder` first estimates $\epsilon(x, x', \Phi)$ with correlated sampling, resulting in $\hat{\epsilon}(x, x', \Phi)$, which estimates $\epsilon(x, x', \Phi)$ well but is also non-differentiable. Then, `DP-Finder` transforms $\hat{\epsilon}(x, x', \Phi)$ to a differentiable function $\hat{\epsilon}^d(x, x', \Phi)$. This is then passed to a numerical optimizer to find a maximal counterexample, with respect to $\hat{\epsilon}^d(x, x', \Phi)$. When a counterexample is returned from the optimizer, `DP-Finder` computes the estimated violation $\hat{\epsilon}(x, x', \Phi)$, with correlated sampling, or computes the exact violation $\epsilon(x, x', \Phi)$, with symbolic solvers.

violations. In this work, we address both challenges and present an effective approach for discovering ϵ-DP violations.

Main Insights. Our approach to finding privacy violations relies on two main insights. First, we can estimate ϵ using correlated sampling with relatively few samples that still provide a good estimate for ϵ. To determine the quality of the estimate, we develop a heuristic inspired by the *central limit theorem* (CLT) and leveraging Hinkley's theorem on the ratio of Gaussian random variables [19]. Second, to search through the sparse space of privacy violations, we define a differentiable surrogate function that allows us to leverage numerical optimization methods.

DP-Finder. Based on the above insights, we present a system, called `DP-Finder`. The goal of `DP-Finder` is to find a triple (x, x', Φ) witnessing the largest possible privacy violation. The values x and x' are inputs to the randomized algorithm, while Φ is a possible set of outputs. Fig. 1 illustrates the high-level approach of `DP-Finder`. The leftmost plot illustrates the search space of `DP-Finder`: it shows the privacy violation ϵ as a function of x, x', and Φ. Since it is computationally prohibitive to directly maximize $\epsilon(x, x', \Phi)$, `DP-Finder` estimates $\epsilon(x, x', \Phi)$ with correlated sampling, denoted $\hat{\epsilon}(x, x', \Phi)$. To ensure that the estimate is precise, `DP-Finder` determines the number of required samples to achieve a given target precision. In general, $\hat{\epsilon}(x, x', \Phi)$ is non-differentiable (illustrated by the discontinuities in its graph). Thus, in the next step, `DP-Finder` makes $\hat{\epsilon}(x, x', \Phi)$ differentiable through a set of rewrite rules, resulting in a differentiable violation estimate, denoted $\hat{\epsilon}^d(x, x', \Phi)$. Then, `DP-Finder` uses an off-the-shelf numerical optimizer to find a triple (x, x', Φ) with a high privacy violation, with respect to $\hat{\epsilon}^d(x, x', \Phi)$. Finally, `DP-Finder` computes the true privacy violation $\epsilon(x, x', \Phi)$ for the final triple (x, x', Φ) using exact symbolic probabilistic solvers (e.g., PSI [16]), if they succeed (PSI succeeded in all of our experiments), otherwise we output $\hat{\epsilon}(x, x', \Phi)$.

We implemented a prototype of `DP-Finder` and evaluated it on a number of randomized algorithms. To the best of our knowledge, `DP-Finder` is the first system capable of automatically estimating differential privacy lower bounds using a general method, applicable to a wide range of algorithms. Our results demonstrate that, often, the lower bounds discovered by `DP-Finder` are close to the known upper bounds (implying tightness). For example, we show that the `noisyMax` algorithm, which was proven to satisfy 10%-DP [26], is not 9.9%-DP. This implies that we can characterize the exact ϵ as lying in the tight interval [9.9%, 10%]. In few cases, we compute a lower bound that is further from the respective upper bound. For example, for the `AboveThreshold` algorithm, which is known to be (at least) 45%-DP [26], for arrays of size 4, we were only able to find 17.3%-DP violations. This suggests that the known upper bound can potentially be improved or, alternatively, further research is needed to discover better lower bounds.

Main Contributions. To summarize, our main contributions are:

- An approach that estimates privacy violations through correlated sampling, along with a confidence interval (Sec. 4).
- A transformation which translates the non-differentiable estimation into a differentiable one, enabling use of numerical optimizers for finding privacy violations of ϵ-DP (Sec. 5).
- An implementation[1] and evaluation on a number of randomized algorithms, showing that our approach is effective in discovering useful privacy violations of ϵ (Sec. 6).

2 PROBLEM STATEMENT

We address the problem of finding a *counterexample* to ϵ-DP, for large ϵ. In the following, we introduce background terms that help precisely define the notion of ϵ-DP counterexamples.

Randomized Algorithms. We consider randomized algorithms that obscure their inputs by adding *noise* to computation steps. The noise is a random variable with a given distribution, e.g., the Laplace distribution. For example, consider the above threshold randomized algorithm `AT` (Fig. 2), simplified from Lyu et al. [26]. It is parameterized by a threshold `T`, and, for an array `x`, it returns a Boolean array `y` whose ith entry indicates whether `x[i]` exceeds `T`. To reduce information leakage of `x`, it adds noise to the threshold

[1]The implementation is available at https://github.com/eth-sri/dp-finder

```
def AT(x):
  ρ = Lap(20)
  for i =  1 to k:
    ν[i] = Lap(20)
    if x[i]+ν[i]≥T+ρ:
      y[i] = 1
    else
      y[i] = 0
  return y
```

Figure 2: An instance of the above threshold randomized algorithm.

```
def check_AT,{[0,0]}(x):
    y = AT(x)
    c = [0, 0]
    ret = 1
    for i = 1 to 2:
      ret = ret &&
            y[i] == c[i]
    return ret
```

Figure 3: Attacker's check on AT and $\{[0,0]\}$ consists of running AT and checking inclusion in $\{[0,0]\}$.

and the entries of x. The noise terms are drawn from the Laplace distribution with scale 20. We note that in the original randomized algorithm, the noise terms are drawn from a Laplace distribution which is determined by the target upper bound ϵ (in particular, the distribution scale is $2/\epsilon$). To avoid confusion with this ϵ and a lower bound on the optimal ϵ (which is what DP-Finder is searching for), we instantiate the target upper bound with $\epsilon = 0.1$, resulting in a distribution scale of 20.

Differential Privacy (DP). A (randomized) algorithm $F\colon \mathcal{X} \to \mathcal{Y}$ is ϵ-*differentially private* (ϵ-DP) if for every pair of *neighboring* inputs $x, x' \in \mathcal{X}$, and for every (measurable) set $\Phi \subseteq \mathcal{Y}$, the probabilities of events $F(x) \in \Phi$ and $F(x') \in \Phi$ are closer than a factor of $\exp(\epsilon)$:

$$\Pr\left[F(x) \in \Phi\right] \leq \exp(\epsilon) \Pr\left[F(x') \in \Phi\right].$$

In this work, we focus on algorithms over the real vectors, i.e., $\mathcal{X} = \mathbb{R}^k$, for some k; however, our results extend to other domains, e.g., matrices of real numbers. Additionally, we assume $\mathcal{Y} = \mathbb{R}^l$ or $\mathcal{Y} = \mathcal{D}^l$, for some l and a finite set $\mathcal{D}$. For example, in AT, $\mathcal{Y} = \{0,1\}^k$, i.e., $\mathcal{D} = \{0,1\}$ and $l = k$, implying that the output is a Boolean array with the same size as the input array. If $\mathcal{Y} = \mathcal{D}^l$ (i.e., $\mathcal{Y}$ is discrete), it suffices to only consider individual outputs ($y \in \mathcal{Y}$), which can be captured in our setting through singleton sets $\Phi = \{y\}$.

We next define the concept of a neighborhood. This concept is inspired from databases, in which x and x' are viewed as databases, and they are neighbors if they differ only in a single user's data. Then, if differential privacy holds, the output distribution of F is almost the same for x and x', i.e., adding the differentiating user's data does not affect F's output significantly. Formally, a neighborhood is captured by a binary relation $\mathbf{Neigh} \subseteq \mathcal{X} \times \mathcal{X}$, i.e., inputs x and x' are neighbors if $(x, x') \in \mathbf{Neigh}$. A possible instantiation of $\mathbf{Neigh}$ is the set of all array pairs whose entries differ by at most 1, denoted $\mathbf{Neigh}_{\leq 1}$:

$$(x, x') \in \mathbf{Neigh}_{\leq 1} \Leftrightarrow \forall i \in \{0, \ldots, k-1\}.\ |x_i - x'_i| \leq 1.$$

For example, $([7.4, 4.7], [8.1, 4.6]) \in \mathbf{Neigh}_{\leq 1}$. This is useful when a database contains aggregate data on its users, such as counts of how many users suffer from certain diseases. Then, adding the data of a single user can affect each count by at most one.

Attacker's Check. The set $\Phi \subseteq \mathcal{Y}$ can be interpreted as a *check* on the algorithm's output that is performed by the attacker to gain information. For example, the meaning of $\Phi = \{[0,0]\}$ is that the attacker tries to guess which of two possible inputs was used to produce an observed output of AT, by checking if the output is equal to $[0,0]$. If the probability of outputting $[0,0]$ differs substantially for two inputs, this allows the attacker to learn which of the two inputs was likely provided as input. Formally, the randomized program that checks whether the output of a randomized algorithm $F\colon \mathcal{X} \to \mathcal{Y}$ lies in Φ is denoted by $\mathtt{check}_{F,\Phi}\colon \mathcal{X} \to \{0,1\}$ and is defined as:

$$\mathtt{check}_{F,\Phi}(x) = [F(x) \in \Phi],$$

where $[\cdot]$ denotes the Iverson brackets, returning 1 if $F(x) \in \Phi$, and 0 otherwise. Technically, $\mathtt{check}_{F,\Phi}(x)$ runs F on the input x and then executes additional statements to determine whether $F(x)$ belongs to Φ. Note that $\mathtt{check}_{F,\Phi}(x)$ is randomized simply because it runs the randomized algorithm F. Fig. 3 demonstrates the randomized program $\mathtt{check}_{F,\Phi}(x)$ for $F = \mathtt{AT}$ and $\Phi = \{[0,0]\}$.

ϵ-DP Counterexamples. An ϵ-DP counterexample is a triple (x, x', Φ) which violates ϵ-DP:

$$\Pr\left[F(x) \in \Phi\right] > \exp(\epsilon) \cdot \Pr\left[F(x') \in \Phi\right].$$

Assuming $\Pr\left[F(x') \in \Phi\right] \neq 0$, this is equivalent to

$$\epsilon < \log \frac{\Pr\left[F(x) \in \Phi\right]}{\Pr\left[F(x') \in \Phi\right]}. \tag{1}$$

The *privacy violation* $\epsilon(x, x', \Phi)$ for a triple (x, x', Φ) is the supremum of all ϵ satisfying Eq. (1):

$$\epsilon(x, x', \Phi) := \log \frac{\Pr\left[F(x) \in \Phi\right]}{\Pr\left[F(x') \in \Phi\right]}. \tag{2}$$

Our goal is to find a triple with a large privacy violation, i.e., we want to solve the following optimization problem:

$$\begin{array}{ll} \arg\max_{x, x', \Phi} & \epsilon(x, x', \Phi) \\ \text{s.t. } (x, x') \in \mathbf{Neigh} & \end{array} \tag{3}$$

Note that $\epsilon(x, x', \Phi)$ may be negative, in which case we can swap x and x' to get a positive violation or, alternatively, consider $|\epsilon(x, x', \Phi)|$ (this is the approach taken by our implementation). To avoid clutter, we ignore this in the rest of the paper.

Direct Optimization of $\epsilon(x, x', \Phi)$. A straightforward approach is to try and solve the maximization problem in Eq. (3) directly, by (i) computing $\Pr\left[F(x) \in \Phi\right]$ and $\Pr\left[F(x') \in \Phi\right]$ symbolically and (ii) optimizing $\epsilon(x, x', \Phi)$ exactly. We did this by using PSI [16] for (i) and Mathematica[2] for (ii). Unfortunately, even when only looking for counterexamples with array size 2, Mathematica cannot solve the resulting maximization problem (timeout after 6 hours). In addition, PSI times out on larger array sizes (e.g., array size 4 times out after 6 hours).

[2]https://www.wolfram.com/mathematica

Generalization to (ϵ, δ)-DP. We remark that our problem statement follows the original DP definition [12]. A well-known generalization to ϵ-DP is (ϵ, δ)-DP [13], in which the requirement is:

$$\Pr[F(x) \in \Phi] \leq \exp(\epsilon) \Pr[F(x') \in \Phi] + \delta.$$

Note that for $\delta = 0$, $(\epsilon, 0)$-DP is exactly ϵ-DP.

Our problem statement can be generalized to (ϵ, δ)-DP, where δ need not be 0. In this case, the privacy violation depends on δ:

$$\epsilon_{\delta}(x, x', \Phi) := \log \frac{\Pr[F(x) \in \Phi] - \delta}{\Pr[F(x') \in \Phi]}. \tag{4}$$

Solving this generalized problem with our approach is straightforward, but complicates the presentation. Thus, we follow the original definition from here on.

3 OVERVIEW

In this section, we provide an overview of `DP-Finder` and discuss its applications. Full details are provided in later sections.

Challenges. `DP-Finder` aims to solve the maximization problem (3), which introduces two challenges. The first challenge is that it is hard to compute the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$, for x, x', and Φ, and these need to be evaluated for many intermediate results during optimization. While analytic approaches (e.g., PSI [16]) can be used, they are computationally expensive. Approximating the quantities using random sampling may also incur high costs if the number of required samples is too high.

The second challenge is to efficiently search the space of triples so to find one with a large privacy violation – because the solution space is sparse, random search is inherently ineffective.

Our Approach. We address these challenges in two steps. First, since the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ are hard to compute, we replace them by an estimate based on sampling. We address the crucial decision of how many samples to use during the sampling by computing (heuristic) confidence intervals on our estimates, and increasing the number of samples until the confidence intervals are sufficiently small.

Second, since the resulting optimization goal is not differentiable, we replace it with a differentiable function. This allows us to search for triples with large privacy violations by maximizing the resulting differentiable function using off-the-shelf numerical optimizers.

While the differentiable maximization goal is (naturally) not equivalent to the original maximization goal, in our evaluation, we show that optimizing the differentiable maximization goal produces triples with high privacy violations in practice.

Flow. We now demonstrate our end-to-end approach on an example. Fig. 5 exemplifies the flow of `DP-Finder` on the randomized algorithm `AT`. `DP-Finder` takes as input a randomized algorithm F (e.g., `AT`) and runs N iterations of the pipeline. In each iteration, `DP-Finder` performs a local search, resulting in a triple (x, x', Φ) with two neighboring inputs x, x' and a set of outputs Φ. Eventually, the triple with the highest associated privacy violation is returned.

At the start of each iteration, `DP-Finder` randomly picks a triple of x, x', and Φ (e.g., $x = [7.4, 4.7]$, $x' = [8.1, 4.6]$, and $\Phi = \{[0, 0]\}$ in Fig. 5). Based on the type of Φ, `DP-Finder` then generates the

```
def check¹_AT,{[0,0]}(x):
  ρ = 7.5
  ν[1] = -23.3
  if x[1]+ν[1] ≥ T+ρ:
    y[1] = 1
  else
    y[1] = 0
  ν[2] = 24.3
  if x[2]+ν[2] ≥ T+ρ:
    y[2] = 1
  else
    y[2] = 0
  return y[1] == 0 &&
         y[2] == 0
```

```
def check²_AT,{[0,0]}(x):
  ρ = 38.3
  ν[1] = 35.5
  if x[1]+ν[1] ≥ T+ρ:
    y[1] = 1
  else
    y[1] = 0
  ν[2] = -14.0
  if x[2]+ν[2] ≥ T+ρ:
    y[2] = 1
  else
    y[2] = 0
  return y[1] == 0 &&
         y[2] == 0
```

Figure 4: Samples derived from $\texttt{check}_{\texttt{AT},\{[0,0]\}}$.

corresponding program $\text{check}_{F,\Phi}(x)$.[3] Then, it constructs a differentiable function producing estimates for the privacy violation $\epsilon(x, x', \Phi)$ using the two steps already mentioned, which we further elaborate on next.

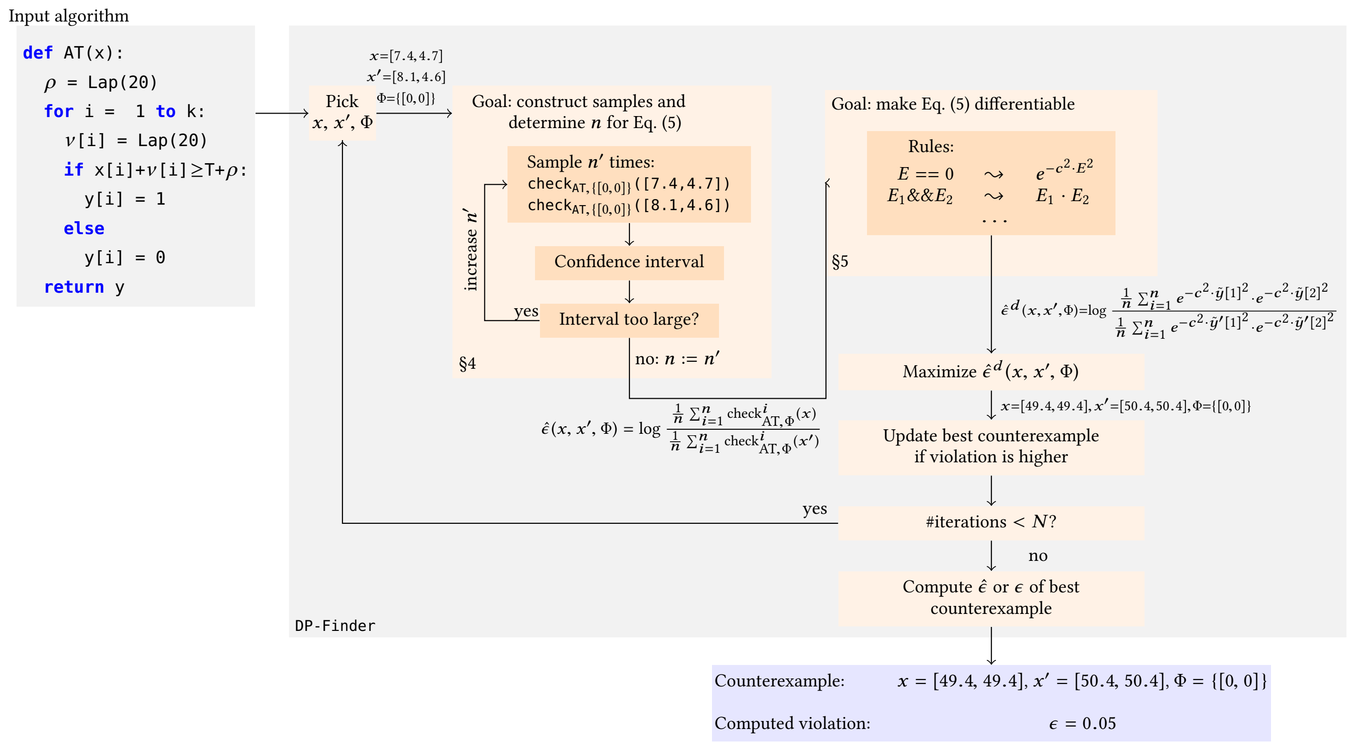
Step 1: Estimating Probabilities by Sampling. `DP-Finder` constructs estimates for the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$. It estimates them using n deterministic programs, denoted $\texttt{check}^1_{F,\Phi}(x), \ldots, \texttt{check}^n_{F,\Phi}(x)$, corresponding to random samples derived from the randomized program $\text{check}_{F,\Phi}(x)$. With these deterministic programs, `DP-Finder` computes an estimate of the privacy violation ϵ for a triple (x, x', Φ) by:

$$\hat{\epsilon}(x, x', \Phi) = \log \frac{\frac{1}{n} \sum_{i=1}^{n} \texttt{check}^i_{F,\Phi}(x)}{\frac{1}{n} \sum_{i=1}^{n} \texttt{check}^i_{F,\Phi}(x')}. \tag{5}$$

Each deterministic program $\texttt{check}^i_{F,\Phi}(x)$ is obtained by unrolling loops in $\texttt{check}_{F,\Phi}$ and fixing the values of the random variables within the randomized program $\texttt{check}_{F,\Phi}(x)$ (i.e., the random variables that appear in the randomized algorithm F), by sampling from their respective distributions. The value of $\text{check}^i_{F,\Phi}(x)$ is the outcome of the attacker's check (for any given input x). That is, it is 1 if $F(x)$ produces an output in Φ using the particular randomness encoded in $\text{check}^i_{F,\Phi}(x)$; otherwise, it is 0. Fig. 4 shows two deterministic programs derived from $\text{check}_{\texttt{AT},\{[0,0]\}}(x)$.

Note that the construction of the deterministic programs is independent of the input. In particular, we assume that the noise distributions do not depend on the input and that loops are bounded. In Sec. 8, we discuss how to extend `DP-Finder` to algorithms whose noise depends on the input. This allows us to *correlate the noise terms* between the samples of $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$. Namely, the programs $\text{check}^i_{F,\Phi}(x)$ and $\text{check}^i_{F,\Phi}(x')$ use the same randomness. This helps to reduce the number of samples n needed to obtain a good estimate. Note, however, that, for $i \neq j$, $\text{check}^i_{F,\Phi}(x)$ and $\text{check}^j_{F,\Phi}(x)$ use independent randomness.

[3]Currently, Φ can be either a singleton set or a box (i.e., an interval generalized to multiple dimensions), but `DP-Finder` can be easily extended to other kinds of sets.


Input algorithm

```
def AT(x):
  ρ = Lap(20)
  for i =  1 to k:
    ν[i] = Lap(20)
    if x[i]+ν[i]≥T+ρ:
      y[i] = 1
    else
      y[i] = 0
  return y
```

Pick x, x', Φ
$x=[7.4,4.7]$
$x'=[8.1,4.6]$
$\Phi=\{[0,0]\}$
Goal: construct samples and determine n for Eq. (5)
Sample n' times:
check$_{\text{AT},\{[0,0]\}}$([7.4,4.7])
check$_{\text{AT},\{[0,0]\}}$([8.1,4.6])
Confidence interval
Interval too large?
yes
increase n'
no: $n := n'$
§4
$\hat{\epsilon}(x, x', \Phi) = \log \frac{\frac{1}{n}\sum_{i=1}^{n}\text{check}^{i}_{\text{AT},\Phi}(x)}{\frac{1}{n}\sum_{i=1}^{n}\text{check}^{i}_{\text{AT},\Phi}(x')}$
Goal: make Eq. (5) differentiable
Rules:
$E == 0 \rightsquigarrow e^{-c^2 \cdot E^2}$
$E_1 \&\& E_2 \rightsquigarrow E_1 \cdot E_2$
...
§5
$\hat{\epsilon}^{d}(x, x', \Phi) = \log \frac{\frac{1}{n}\sum_{i=1}^{n} e^{-c^2 \cdot \tilde{y}[1]^2} \cdot e^{-c^2 \cdot \tilde{y}[2]^2}}{\frac{1}{n}\sum_{i=1}^{n} e^{-c^2 \cdot \tilde{y}'[1]^2} \cdot e^{-c^2 \cdot \tilde{y}'[2]^2}}$
Maximize $\hat{\epsilon}^{d}(x, x', \Phi)$
$x=[49.4,49.4], x'=[50.4,50.4], \Phi=\{[0,0]\}$
Update best counterexample if violation is higher
#iterations $< N$?
yes
no
Compute $\hat{\epsilon}$ or ϵ of best counterexample
DP-Finder
Counterexample: $x = [49.4, 49.4]$, $x' = [50.4, 50.4]$, $\Phi = \{[0, 0]\}$
Computed violation: $\epsilon = 0.05$

Figure 5: DP-Finder takes an algorithm and iteratively samples a triple, transforms its privacy violation to a differentiable function, which is then optimized to find a counterexample with a higher privacy violation than the sampled triple. Finally, it returns the counterexample with the highest privacy violation found.

Given this construction, it remains to pick a suitable number of samples n, which is small enough to avoid costly computations, yet sufficiently large to provide a good estimate for $\epsilon(x, x', \Phi)$. For our purposes, n is sufficiently large if, for the initial triple (x, x', Φ), it results in an estimate with a small *confidence interval* (e.g., smaller than $2 \cdot 10^{-3}$). To find such an n, `DP-Finder` starts from a small n' and gradually draws more samples. Occasionally, `DP-Finder` estimates a (heuristic) confidence interval, and only continues sampling if it is still too large. In our example, `DP-Finder` computes correlated samples for check$_{\text{AT},\{[0,0]\}}$([7.4, 4.7]) and check$_{\text{AT},\{[0,0]\}}$([8.1, 4.6]).

To obtain a confidence interval, we use a heuristic based on the (multivariate) central limit theorem (CLT) and prior work on the ratio of two correlated Gaussian random variables [19]. Note that correlating the random noise is necessary in order to reduce the number of samples. We provide details in Sec. 4.

Step 2: Make the Estimate Differentiable. In the second step, `DP-Finder` translates $\hat{\epsilon}$ from Eq. (5) into a differentiable version $\hat{\epsilon}^{d}(x, x', \Phi)$. To this end, it translates the deterministic programs $\text{check}^{i}_{F,\Phi}(x)$ to differentiable programs using a set of a rewrite rules on its statements. As Boolean expressions and conditionals are the only source of non-differentiability, we transform them to differentiable functions. Our rules have the property that, if the Boolean expression is true, the corresponding differentiable program evaluates to a value close to 1, otherwise it evaluates to a value close to 0. For example, we transform the constraint $x = 0$ to the function $\exp(-c^2 \cdot x^2)$, which for large enough c (we use 50) is very steep. Similarly, we rewrite logical and (&&) to multiplication. Using these and additional rules, `DP-Finder` generates a differentiable program. For example, translating check$^{1}_{\text{AT},\{[0,0]\}}$(`x`) from Fig. 4 results in the differentiable program:

$$\texttt{return}\ \underbrace{\exp(-c^2 \cdot \tilde{y}[1]^2)}_{\texttt{y[1]==0}} \underbrace{\cdot}_{\&\&} \underbrace{\exp(-c^2 \cdot \tilde{y}[2]^2)}_{\texttt{y[2]==0}},$$

where $\tilde{y}[1]$ and $\tilde{y}[2]$ are the differentiable functions corresponding to $y[1]$ and $y[2]$ (omitted here for simplicity). We provide details in Sec. 5.

Finally, `DP-Finder` feeds the resulting expression $\hat{\epsilon}^{d}(x, x', \Phi)$ to a numerical optimizer, which solves the problem

$$\begin{array}{ll} \arg\max_{x, x', \Phi} & \hat{\epsilon}^{d}(x, x', \Phi). \\ \text{s.t. } (x, x') \in \textbf{Neigh} & \end{array}$$

For our example, `DP-Finder` finds the triple with $x = [49.4, 49.4]$, $x' = [50.4, 50.4]$ and $\Phi = \{[0, 0]\}$, with an estimated violation of 0.05 (whose violation is larger than the violation for the initial triple). If this triple has higher privacy violation than the recorded one, `DP-Finder` updates it.

Final Step: Computing the Privacy Violation of the Returned Triple. After completing all of its N iterations, `DP-Finder` returns the triple that induced the highest (estimated) privacy violation. However, it is possible that the estimated privacy violation for that triple is not accurate, due to the approximations (sampling, differentiable estimate) DP-Finder applies.

To mitigate this, DP-Finder computes the exact privacy violation ϵ using an exact solver (PSI [16]) with the triple (x, x', Φ) returned by the optimizer. Note that in this case, we run the solver with concrete inputs, not symbolic ones. If the solver does not complete within a given timeout, DP-Finder estimates the privacy violation using Eq. (5), with a high number of samples, yielding a guaranteed confidence interval. Then, it returns this triple, along with its (estimated) privacy violation. In our evaluation, PSI never times out.

Applications of DP-Finder. DP-Finder is a complementary approach to prior works that prove ϵ-DP of particular algorithms. With DP-Finder, one can prove, using explicit counterexamples, that the proved ϵ is the smallest possible. Alternatively, if DP-Finder cannot find a triple whose violation is close to the proven bound, this suggests that it may be possible to tighten the bound.

Another application is to use DP-Finder to find errors in ϵ-DP proofs. For example, the *generalized private threshold testing* algorithm was believed to be private, but follow-up work disproved it using counterexamples [9]. With DP-Finder, it is possible to come up with new counterexamples, thereby providing a best-effort validation to new upper bounds. In fact, even without proofs, DP-Finder can be useful for studying the privacy of algorithms. For example, consider a (non-expert) user that tweaks a privacy algorithm and wants to test whether the changes made have significantly affected privacy. While DP-Finder provides no guarantees that the privacy violation found is really the largest possible, it is the only existing general framework that can test algorithms for differential privacy.

We also believe that DP-Finder can be used by attackers, depending on the attacker model. In particular, an attacker can use DP-Finder to find input pairs which leak a lot of information. Given enough power, the attacker may stir the inputs towards those that leak information. We leave the study of attacks that can use DP-Finder to future work.

4 ESTIMATION OF PRIVACY VIOLATION WITH CONFIDENCE

In this section, we present our approach for estimating the privacy violation $\epsilon(x, x', \Phi)$ for a given triple (x, x', Φ). We begin by explaining our sampling approach. Then, we explain how to determine a confidence interval for our estimate and how to reduce the sampling effort.

4.1 Estimation of Privacy Violations

We now explain how we estimate $\epsilon(x, x', \Phi)$ as a closed-form function of (x, x', Φ).

Recall that in Eq. (2) we defined:

$$\epsilon(x, x', \Phi) := \log \frac{\Pr[F(x) \in \Phi]}{\Pr[F(x') \in \Phi]}.$$

We want to estimate the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ using sampling, yielding an estimate $\hat{\epsilon}(x, x', \Phi)$ of $\epsilon(x, x', \Phi)$.

Estimating Probabilities. To estimate $\Pr[F(x) \in \Phi]$, we sample from a random variable $S \in \{0, 1\}$ defined by $S := \text{check}_{F,\Phi}(x)$, meaning that S is 1 if $F(x) \in \Phi$, and 0 otherwise. Because S follows a Bernoulli distribution, its expectation is the probability we want to estimate:

$$\mathbb{E}[S] = \Pr[S = 1] = \Pr[F(x) \in \Phi].$$

Denoting the samples of S by $S_1, \ldots, S_n$, we estimate $\Pr[F(x) \in \Phi]$ by:

$$\Pr[F(x) \in \Phi] \approx \widehat{\Pr}[F(x) \in \Phi] = \tfrac{1}{n} \textstyle\sum_{i=1}^{n} S_i.$$

Correlating Random Choices. Instead of directly sampling S, we first randomly sample deterministic programs $\text{check}^i_{F,\Phi}$ for $i = 1, \ldots, n$. The programs are defined by randomly sampling values for the random variables within F from their respective distributions. The random variables are then replaced by fixed values to obtain each of the programs $\text{check}^i_{F,\Phi}$. The output of $\text{check}^i_{F,\Phi}(x)$ is 1 or 0 depending on whether $F(x)$ produces an output in Φ, using the particular fixed randomness encoded into $\text{check}^i_{F,\Phi}$.

This allows us to *correlate* the outputs of $\text{check}_{F,\Phi}(x)$ and $\text{check}_{F,\Phi}(x')$, by running the latter using the *same* fixed randomness as the former, i.e., by estimating $\Pr[F(x') \in \Phi]$ using $\text{check}^i_{F,\Phi}(x')$:

$$\Pr[F(x') \in \Phi] \approx \widehat{\Pr}[F(x') \in \Phi] = \tfrac{1}{n} \textstyle\sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x').$$

In Sec. 4.5, we show how to leverage this correlation to reduce the number of samples.

Estimating $\epsilon(x, x', \Phi)$. Overall, we compute the estimate $\hat{\epsilon}(x, x', \Phi)$ as follows:

$$\hat{\epsilon}(x, x', \Phi) = \log \frac{\widehat{\Pr}[F(x) \in \Phi]}{\widehat{\Pr}[F(x') \in \Phi]} = \log \frac{\frac{1}{n} \sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x)}{\frac{1}{n} \sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x')}. \quad (6)$$

In the following, we often denote $\text{check}^i_{F,\Phi}(x)$ by S_i, and $\text{check}^i_{F,\Phi}(x')$ by S'_i.

4.2 Challenge: Determining Sampling Effort

To obtain a good estimate for the privacy violation $\epsilon(x, x', \Phi)$, it is crucial to select a suitable number of samples n. Clearly, we can obtain estimates with better probabilistic guarantees by increasing the number of samples. However, this induces a higher computational cost in the optimization step, increasing DP-Finder's running time.

Accordingly, we want to pick the smallest number of samples n, for which the error $|\epsilon(x, x', \Phi) - \hat{\epsilon}(x, x', \Phi)|$ is small enough. Given a target error bound Δ_ϵ that should hold with high probability, we therefore want to select n as small as possible.

Computing Confidence Intervals for $\epsilon(x, x', \Phi)$. While computing a number of samples n that achieves a specific target error bound directly is non-trivial, it is easier to compute the error bound Δ_ϵ that results from picking a specific n. Assuming we can solve the latter, we can find an appropriate n by gradually drawing samples, checking the resulting error bound, and increasing n (e.g., using exponential search) while the error is too large.

In the following sections, we show three approaches to computing an error bound based on a given n. We express this bound in terms of a *confidence interval* for $\epsilon(x, x', \Phi)$: we want to find Δ_ϵ such that

$$\epsilon(x, x', \Phi) \in [\hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon, \hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon]$$

with probability at least $1 - \alpha$, where α is a small constant. The constant $1 - \alpha$ is also called the *confidence*.

Overview of Approaches. We first show how to derive confidence intervals with strong probabilistic guarantees using Hoeffding's inequality (Sec. 4.3). This is the only approach providing probabilistic guarantees, however, it results in confidence intervals that are empirically larger than necessary. As a consequence, applying Hoeffding's inequality results in a large number of samples, especially for small probabilities, as demonstrated by Fig. 6.

To reduce the required number of samples, we trade the guarantees of the former approach against more efficiency, by estimating the confidence interval using a *heuristic* inspired by the central limit theorem (CLT, Sec. 4.4). While this already reduces the required number of samples, it sets the stage for a drastic reduction that we achieve by taking into account the correlation of the random samples $\text{check}^i_{F,\Phi}(x)$ and $\text{check}^i_{F,\Phi}(x')$ using the multidimensional CLT (M-CLT, Sec. 4.5).

Fig. 6 shows the number of samples required to achieve a fixed absolute error of $\Delta_\epsilon = 2 \cdot 10^{-3}$ with (approximate) confidence 90%, as a function of the probability being estimated (which is the most relevant factor influencing the required number of samples), with each of the three presented approaches. For example, the figure shows that estimating $\epsilon(x, x', \Phi)$ to a precision of $2 \cdot 10^{-3}$, for $\Pr[F(x) \in \Phi] = 0.1$ (i.e., 10%) and $\Pr[F(x') \in \Phi] \approx \Pr[F(x) \in \Phi]$, requires almost 10^8 samples with the CLT approach. Fig. 6 demonstrates that the M-CLT approach consistently outperforms the other two approaches, regardless of the probability being estimated.

Combining Approaches. `DP-Finder` uses the approach in Sec. 4.5 to estimate the confidence interval, because this is the most efficient approach. Recall that to mitigate possible imprecisions of $\hat{\epsilon}(x, x', \Phi)$, `DP-Finder` recomputes the privacy violation of the returned triple with a exact solver (PSI). If PSI times out, `DP-Finder` can combine Sec. 4.5 with the approach in Sec. 4.3 to get the best of both worlds: It can first use `DP-Finder` to search for a triple (x, x', Φ), using the heuristic confidence interval during the search (empirically, this works well). To estimate the obtained $\epsilon(x, x', \Phi)$ with strong guarantees, it can then use a higher number of samples, such that a small confidence interval can be derived using Hoeffding's inequality.

4.3 Approach 1: Guaranteed Confidence Intervals based on Hoeffding's inequality

We can use Hoeffding's inequality to estimate a confidence interval for $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$, which in turn gives us a confidence interval for:

$$\epsilon(x, x', \Phi) = \log \frac{\Pr[F(x) \in \Phi]}{\Pr[F(x') \in \Phi]}.$$

Concretely, recall $S := \text{check}_{F,\Phi}(x)$ and its expectation $\mathbb{E}[S] = \Pr[F(x) \in \Phi]$, and the samples $S_1, \ldots, S_n$ are from S.

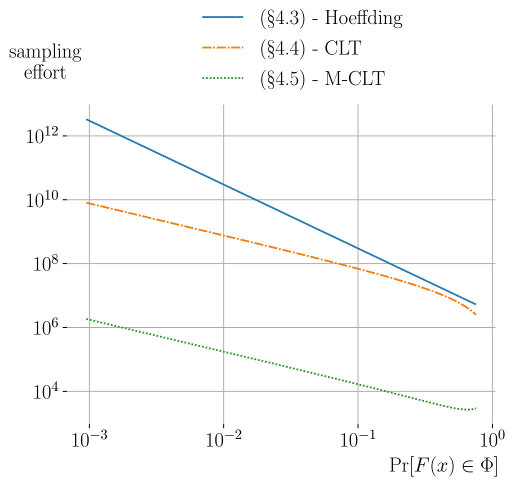


Figure 6: Number of samples needed to estimate $\epsilon(x, x', \Phi)$ up to an error of $2 \cdot 10^{-3}$ with empirical confidence of 90% as a function of $\Pr[F(x) \in \Phi]$. Here, we assume that $\Pr[F(x) \in \Phi] \approx \Pr[F(x') \in \Phi]$ and that the correlation coefficient of $\text{check}_{F,\Phi}(x)$ and $\text{check}_{F,\Phi}(x')$ is $\rho = 0.999$.

Intuitively, Hoeffding's inequality states that the probability that the two values $\frac{1}{n}\sum_{i=1}^n S_i$ and $\mathbb{E}[S]$ are further apart than a constant factor is exponentially small in the number of samples n.

Theorem 4.1 (Hoeffding's inequality [20]). *Let S be a Bernoulli distribution, and $S_1, \ldots, S_n \in \{0, 1\}$ be independent samples from S. Let $p = \mathbb{E}[S] \in [0, 1]$ and $\Delta > 0$. Then,*

$$\Pr\left[\left|p - \frac{1}{n}\sum_{i=1}^{n} S_i\right| \geq \Delta\right] \leq 2\exp\left(-2n\Delta^2\right).$$

Required Confidence. To compute an interval for ϵ with confidence $1 - \alpha$, we apply Hoeffding's inequality for $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ (separately) to compute intervals with confidence $1 - \alpha/2$ for each.

Using a confidence of $1 - \alpha/2$ for both intervals allows us to conclude that individually, each confidence interval fails to contain its value ($\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$, respectively) with a probability of $\alpha/2$. According to the union bound, the probability that either interval fails to contain its value is $\alpha/2 + \alpha/2$, resulting in a confidence of $1 - \alpha$ that both intervals contain their respective value simultaneously.

Applying Hoeffding's Inequality. To obtain an interval $[\widehat{\Pr}[F(x) \in \Phi] - \Delta, \widehat{\Pr}[F(x) \in \Phi] + \Delta]$ for $\Pr[F(x) \in \Phi]$ with confidence $1 - \alpha/2$, we define Δ as:

$$\Delta := \sqrt{\frac{\log(\alpha/4)}{-2n}}$$

For this choice of Δ, according to Hoeffding's inequality, the probability that the confidence interval does not contain $\Pr[F(x) \in \Phi]$ is at most $2\exp\left(-2n\Delta^2\right) = \alpha/2$.

We compute the confidence interval for $\Pr[F(x') \in \Phi]$ analogously.

Bounds on $\epsilon(x, x', \Phi)$. Assuming the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ lie in their respective intervals (which simultaneously happens with confidence $1 - \alpha$), we can use these intervals to derive a lower and an upper bound on $\epsilon(x, x', \Phi)$:

$$\underbrace{\log \frac{\widehat{\Pr}[F(x) \in \Phi] - \Delta}{\widehat{\Pr}[F(x') \in \Phi] + \Delta}}_{l} \le \epsilon(x, x', \Phi) \le \underbrace{\log \frac{\widehat{\Pr}[F(x) \in \Phi] + \Delta}{\widehat{\Pr}[F(x') \in \Phi] - \Delta}}_{u}.$$

To get an interval $[\hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon, \hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon]$ for $\epsilon(x, x', \Phi)$ with confidence $1 - \alpha$, we need to ensure that $\hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon \le l$ and $\hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon \le u$. A conservative choice for Δ_ϵ that satisfies these constraints is $\Delta_\epsilon = u - l$.

We note that in general, Δ_ϵ (the width of the confidence interval for $\epsilon(x, x', \Phi)$) is not the same as Δ (the width of the confidence interval for $\Pr[F(x) \in \Phi]$).

Example. We next illustrate this computation, using $n = 10^7$ samples to estimate the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$, targeting a confidence of $1 - \alpha = 1 - 0.1\%$ for $\epsilon(x, x', \Phi)$.

We can compute $\Delta = \sqrt{\frac{\log(\alpha/4)}{-2 \cdot n}} \approx 0.06\%$, without any information about the algorithm or the distribution of the samples $S_1, \ldots, S_n$. Now, assume that for our (x, x', Φ), we get $\widehat{\Pr}[F(x) \in \Phi] \approx 3.24\%$ and $\widehat{\Pr}[F(x') \in \Phi] \approx 3.04\%$. By Hoeffding's inequality, $\Pr[F(x) \in \Phi] \in [3.24\% - \Delta, 3.24\% + \Delta]$ with a confidence of 0.05%. Analogously, $\Pr[F(x') \in \Phi] \in [3.04\% - \Delta, 3.04\% + \Delta]$ with the same confidence.

Now that we have computed confidence intervals for both probabilities, we want to compute a confidence interval for $\epsilon(x, x', \Phi)$. By the union bound, we know with a confidence of $1 - \alpha$ that simultaneously, $\Pr[F(x) \in \Phi] \in [3.24\% - \Delta, 3.24\% + \Delta]$ *and* $\Pr[F(x') \in \Phi] \in [3.04\% - \Delta, 3.04\% + \Delta]$. Based on this, we derive that:

$$\underbrace{\log \frac{3.24\% - 0.06\%}{3.04\% + 0.06\%}}_{\approx 0.024} \le \epsilon(x, x', \Phi) \le \underbrace{\log \frac{3.24\% + 0.06\%}{3.04\% - 0.06\%}}_{\approx 0.103}$$

Hence, $\Delta_\epsilon = 0.103 - 0.024 = 0.079$, meaning that with confidence $1 - 0.1\%$, $\epsilon(x, x', \Phi) \in [\hat{\epsilon}(x, x', \Phi) - 0.079, \hat{\epsilon}(x, x', \Phi) + 0.079]$, for $\hat{\epsilon}(x, x', \Phi) \approx \log \frac{3.24\%}{3.04\%} \approx 6.4\%$.

Discussion. Recall that ultimately, we want to find the smallest n that guarantees a given error bound Δ_ϵ on $\epsilon(x, x', \Phi)$. To achieve this goal, we could alternatively derive a closed-form solution for the required number of samples n to achieve a given error Δ_ϵ with confidence $1 - \alpha$ using Hoeffding's inequality directly (instead of gradually increasing the number of samples n). However, since `DP-Finder` applies the approach described in Sec. 4.5, for which we do not have such a closed-form solution, we do not further elaborate on this possibility.

We note that the confidence intervals on the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ provided by Hoeffding's inequality are agnostic of the algorithm under consideration, and of the samples produced from the algorithm. While this is desirable to achieve strong guarantees, it means that this approach cannot profit from additional information on the variance or correlation of the samples $S_1, \ldots, S_n$ and $S'_1, \ldots, S'_n$, ultimately resulting in a conservatively high number of samples n.

4.4 Approach 2: Heuristic Confidence Intervals based on Central Limit Theorem

The approach based on Hoeffding's inequality provides guaranteed confidence intervals, but the width of the intervals is pessimistically large. In this section, we show how to apply a heuristic inspired by the central limit theorem to obtain an approximate confidence interval which is slightly more narrow. Ultimately, this means that we can achieve an (approximate) confidence interval of a certain width with fewer samples. This first heuristic approach sets the stage for our second, dramatically improved heuristic, which also takes into account that our probability estimates are correlated.

Central Limit Theorem. Intuitively, the central limit theorem states that for large n, $\widehat{\Pr}[F(x) \in \Phi] := \frac{1}{n}\sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x)$ is approximately distributed according to a Gaussian distribution, with mean $\mathbb{E}\left[\text{check}_{F,\Phi}\right]$ and variance $\frac{1}{n}\text{Var}\left[\text{check}_{F,\Phi}(x)\right]$. Note that because $\mathbb{E}\left[\text{check}_{F,\Phi}(x)\right] = \Pr[F(x) \in \Phi]$, we have in particular that as $n \to \infty$, $\frac{1}{n}\sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x) \to \Pr[F(x) \in \Phi]$. We next state the CLT.

Theorem 4.2 (Central limit theorem, Prop. 2.18 from [34]). *Let $S_1, \ldots, S_n$ be independent samples from a distribution S over $\mathbb{R}$. Let $\mathbb{E}[S] \in \mathbb{R}$ be the expectation of S, and $\text{Var}[S] \in \mathbb{R}$ the variance of S. Then, as $n \to \infty$ the distribution of $\frac{1}{n}\sum_{i=1}^{n} S_i$ converges to a Gaussian distribution with mean $\mathbb{E}[S]$ and variance $\frac{1}{n}\text{Var}[S]$:*

$$\frac{1}{n}\sum_{i=1}^{n} S_i \xrightarrow[n\to\infty]{d} N\left(\mathbb{E}[S], \frac{1}{n}\text{Var}[S]\right). \tag{7}$$

We note that our version of the CLT deviates from the standard presentation to simplify its application in our case. First, the most common version of the CLT (e.g., [34], Prop. 2.18) states that

$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} S_i - \mathbb{E}[S]\right) \xrightarrow[n\to\infty]{d} N(0, \text{Var}[S]).$$

Instead, we abuse notation by using n within the limiting distribution $N\left(\mathbb{E}[S], \frac{1}{n}\text{Var}[S]\right)$. Second, $\{S_1, \ldots, S_n\}$ is often referred to as a single sample, with sample size n. Instead, we refer to each individual S_i as a sample, and say $\{S_1, \ldots, S_n\}$ consists of n samples.

Heuristic Inspired by the CLT. In the following, we (heuristically) assume that the average $\frac{1}{n}\sum_{i=1}^{n} S_i$ follows the limiting distribution $N\left(\mathbb{E}[S], \frac{1}{n}\text{Var}[S]\right)$, even for $n < \infty$.

Of course, this assumption does not hold strictly mathematically speaking (e.g., for $n = 1$, it states that S follows a Gaussian instead of a Bernoulli distribution). However, for large n (e.g., $n = 10^3$), $\frac{1}{n}\sum_{i=1}^{n} S_i$ follows $N\left(\mathbb{E}[S], \frac{1}{n}\text{Var}[S]\right)$ almost exactly. Empirically, this assumption only introduces negligible imprecisions, enabling us to produce tight confidence intervals in practice.

If we assume that the distribution of $\widehat{\Pr}[F(x) \in \Phi] = \frac{1}{n}\sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x)$ has converged to its limiting distribution according to the CLT, we obtain

$$\underbrace{\widehat{\Pr}[F(x) \in \Phi]}_{\frac{1}{n}\sum_{i=1}^{n} \text{check}^i_{F,\Phi}(x)} \sim N\Big(\underbrace{\Pr[F(x) \in \Phi]}_{\mathbb{E}[\text{check}_{F,\Phi}(x)]}, \frac{1}{n}\text{Var}\left[\text{check}_{F,\Phi}(x)\right]\Big).$$

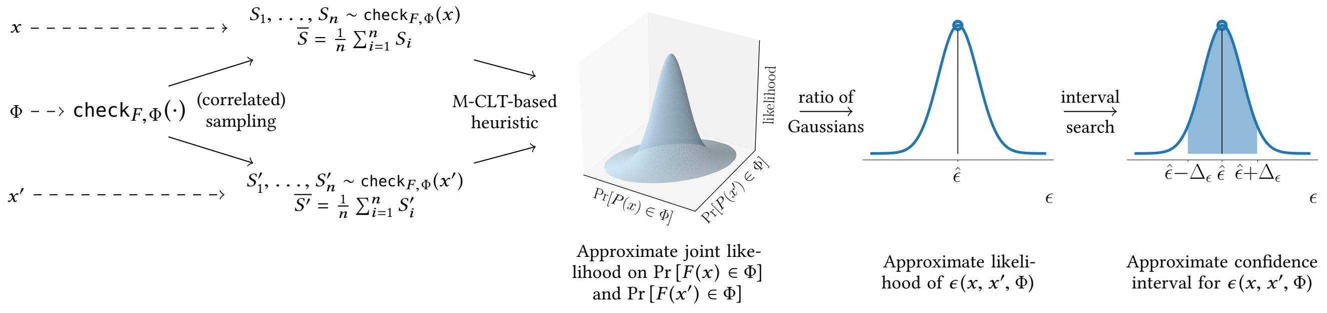

x
Φ
$\mathtt{check}_{F,\Phi}(\cdot)$
x'
(correlated) sampling
$S_1,\ldots,S_n \sim \mathtt{check}_{F,\Phi}(x)$
$\overline{S} = \frac{1}{n}\sum_{i=1}^n S_i$
$S'_1,\ldots,S'_n \sim \mathtt{check}_{F,\Phi}(x')$
$\overline{S'} = \frac{1}{n}\sum_{i=1}^n S'_i$
M-CLT-based heuristic
likelihood
$\Pr[P(x) \in \Phi]$
$\Pr[P(x') \in \Phi]$
Approximate joint likelihood on $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$
ratio of Gaussians
$\hat{\epsilon}$
ϵ
Approximate likelihood of $\epsilon(x,x',\Phi)$
interval search
$\hat{\epsilon}-\Delta_\epsilon$ $\hat{\epsilon}$ $\hat{\epsilon}+\Delta_\epsilon$
ϵ
Approximate confidence interval for $\epsilon(x,x',\Phi)$

Figure 7: Steps for deriving a heuristic confidence interval for violations based on samples $S_1,\ldots,S_n$ and $S'_1,\ldots,S'_n$.

Computing the Likelihood of $\Pr[F(x) \in \Phi]$. Based on the distribution of $\widehat{\Pr}[F(x) \in \Phi]$, we conclude that observing $\widehat{\Pr}[F(x) \in \Phi]$ induces a likelihood on $\Pr[F(x) \in \Phi]$ given by

$$\Pr[F(x) \in \Phi] \sim N\left(\widehat{\Pr}[F(x) \in \Phi], \frac{1}{n}\mathrm{Var}\left[\mathrm{check}_{F,\Phi}(x)\right]\right). \quad (8)$$

For details on the derivation of this likelihood, see App. A. We note that while mathematically, $\Pr[F(x) \in \Phi]$ is a constant, after observing the estimate $\widehat{\Pr}[F(x) \in \Phi]$, some values of $\Pr[F(x) \in \Phi]$ are more likely than others, thus inducing a distribution on $\Pr[F(x) \in \Phi]$, which we call the *likelihood* of $\Pr[F(x) \in \Phi]$.

Computing the Confidence Interval. From Eq. (8), we can derive an interval for $\Pr[F(x) \in \Phi]$ with confidence $1-\alpha/2$. The confidence interval is $[\widehat{\Pr}[F(x) \in \Phi] - \Delta, \widehat{\Pr}[F(x) \in \Phi] + \Delta]$, for $\Delta := -\sigma \cdot \Phi^{-1}\left(\frac{\alpha/2}{2}\right)$. Here, Φ^{-1} is the inverse of the cumulative distribution function of a Gaussian distribution with mean 0 and standard deviation 1, and σ is the standard deviation of the Gaussian distribution in Eq. (8). At runtime, `DP-Finder` does not have access to the standard deviation $\sigma := \sqrt{\frac{1}{n}\mathrm{Var}\left[\mathrm{check}_{F,\Phi}(x)\right]}$. Instead, it estimates σ empirically, computing Δ according to

$$\Delta := -\sqrt{\frac{1}{n}\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x)\right]} \cdot \Phi^{-1}\left(\frac{\alpha/2}{2}\right),$$

where $\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x)\right]$ is the empirical variance of $\mathrm{check}_{F,\Phi}(x)$ (the definition of empirical variance is given in App. B).

Bounds on $\epsilon(x,x',\Phi)$. As in Sec. 4.3, we compute confidence intervals for $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$, which jointly hold with confidence $1-\alpha$ (again due to the union bound). Then, we can derive bounds on $\epsilon(x,x',\Phi)$, also as in Sec. 4.3.

Example. We illustrate how to use this approach when estimating $\epsilon(x,x',\Phi)$ for some (x,x',Φ). We note that unlike for Hoeffding's inequality, Δ depends on the empirical variance of $\mathrm{check}_{F,\Phi}(x)$, and hence on the samples $S_1,\ldots,S_n$. Thus, we cannot compute Δ before sampling from $\mathrm{check}_{F,\Phi}(x)$.

Assume that for our (x,x',Φ), we get $\widehat{\Pr}[F(x) \in \Phi] \approx 3.24\%$ and $\widehat{\Pr}[F(x') \in \Phi] \approx 3.04\%$. In addition, the empirical variance of $\mathrm{check}_{F,\Phi}(x)$ is $\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x)\right] \approx 3.13\%$ and likewise, $\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x')\right] \approx 2.95\%$ (computed according to App. B).

Based on Eq. (8), we derive an approximate likelihood for $\Pr[F(x) \in \Phi]$, given by $\Pr[F(x) \in \Phi] \sim N\left(3.24\%, \frac{1}{n}3.13\%\right)$. Targeting an overall confidence of $1-\alpha = 1-0.1\%$ for $\epsilon(x,x',\Phi)$, we derive an interval $[3.24\%-\Delta, 3.24\%+\Delta]$ for $\Pr[F(x) \in \Phi]$ with confidence $1-\alpha/2$, by computing $\Delta = -\sqrt{\frac{1}{n}3.13\%} \cdot \Phi^{-1}\left(\frac{\alpha/2}{2}\right) \approx 0.02\%$. An analogous computation yields an interval for $\Pr[F(x') \in \Phi]$ with confidence $1-\alpha/2$, given by $[3.04\%-0.019\%, 3.04\%+0.019\%]$.

Exactly as in Sec. 4.3, using both confidence intervals, we derive (with confidence $1-\alpha$), that

$$\underbrace{\log\frac{3.24\%-0.02\%}{3.04\%+0.019\%}}_{\approx 0.051} \leq \epsilon(x,x',\Phi) \leq \underbrace{\log\frac{3.24\%+0.02\%}{3.04\%-0.019\%}}_{\approx 0.076}.$$

Hence, $\Delta_\epsilon = 0.076 - 0.051 = 0.025$, meaning that with confidence $1-0.1\%$, $\epsilon(x,x',\Phi) \in [\hat{\epsilon}(x,x',\Phi) - 0.025, \hat{\epsilon}(x,x',\Phi) + 0.025]$, for $\hat{\epsilon}(x,x',\Phi) \approx \log\frac{3.24\%}{3.04\%} \approx 6.4\%$.

4.5 Approach 3: Heuristic Confidence Intervals based on Multidimensional CLT

We next explain how to improve the previous approach, which computed the two confidence intervals separately. Now, we show how to reduce the number of required samples, by leveraging that $S_i = \mathrm{check}^i_{F,\Phi}(x)$ and $S'_i = \mathrm{check}^i_{F,\Phi}(x')$ are correlated (i.e., derived from a joint distribution).

We recall that `DP-Finder` generates $\mathrm{check}^i_{F,\Phi}(x)$ and $\mathrm{check}^i_{F,\Phi}(x')$ based on the same randomness, resulting in high correlation between the samples. We empirically observed correlations as high as $\rho = 0.999$. This can drastically decrease the required number of samples, as illustrated in Fig. 6.

Fig. 7 provides an overview of this approach. On a high level, the samples $(S_i, S'_i) = \left(\mathrm{check}^i_{F,\Phi}(x), \mathrm{check}^i_{F,\Phi}(x')\right)$ originate from a joint distribution $\boldsymbol{S}$ over $\mathbb{R}^2$, which has (component-wise) mean $\mathbb{E}[\boldsymbol{S}] \in \mathbb{R}^2$ given by $\mathbb{E}[\boldsymbol{S}] = \left(\Pr[F(x) \in \Phi], \Pr[F(x') \in \Phi]\right)$.

Observing the estimates $\left(\widehat{\Pr}[F(x) \in \Phi], \widehat{\Pr}[F(x') \in \Phi]\right)$ computed according to $\left(\frac{1}{n}\sum_{i=1}^n S_i, \frac{1}{n}\sum_{i=1}^n S'_i\right)$ induces a likelihood on $\left(\Pr[F(x) \in \Phi], \Pr[F(x') \in \Phi]\right)$ (depicted as a multivariate Gaussian distribution in Fig. 7). From this, we can derive a likelihood on $\epsilon(x,x',\Phi)$ (see Fig. 7), and finally compute a confidence interval for $\epsilon(x,x',\Phi)$ (depicted as the shaded area in Fig. 7).

Multidimensional Central Limit Theorem (M-CLT). To analyze the effect of correlation on the size of the confidence interval, we consider the central limit theorem for multivariate distributions:

Theorem 4.3 (Multidimensional central limit theorem, Prop. 2.18 from [34]). *Let $\boldsymbol{S_1}, \dots, \boldsymbol{S_n} \in \mathbb{R}^k$ be independent samples from a distribution $\boldsymbol{S}$ over vectors. Let $\mathbb{E}[\boldsymbol{S}] \in \mathbb{R}^k$ be the (component-wise) expectation of $\boldsymbol{S}$, and $\mathrm{Cov}[\boldsymbol{S}] \in \mathbb{R}^{k \times k}$ the covariance matrix of $\boldsymbol{S}$. Then, as $n \to \infty$ the distribution of $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{S_i}$ converges to a multivariate Gaussian distribution with mean $\mathbb{E}[\boldsymbol{S}] \in \mathbb{R}^k$ and covariance matrix $\frac{1}{n}\mathrm{Cov}[\boldsymbol{S}]$:*

$$\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{S_i} \xrightarrow[n \to \infty]{d} N\left(\mathbb{E}[\boldsymbol{S}], \frac{1}{n}\mathrm{Cov}[\boldsymbol{S}]\right).$$

As in the previous section, our version of the CLT deviates from the standard presentation, by (i) using n within the limiting distribution and (ii) referring to each individual vector $\boldsymbol{S_i} \in \mathbb{R}^k$ as a sample.

Heuristic Inspired by the M-CLT. Analogously to the previous section, we apply a heuristic inspired by the M-CLT ($k = 2$), treating the joint distribution of $\widehat{\Pr}[F(x) \in \Phi]$ and $\widehat{\Pr}[F(x') \in \Phi]$ as having converged to the limiting distribution:

$$\begin{pmatrix} \widehat{\Pr}[F(x) \in \Phi] \\ \widehat{\Pr}[F(x') \in \Phi] \end{pmatrix} \sim N\left(\begin{pmatrix} \Pr[F(x) \in \Phi] \\ \Pr[F(x') \in \Phi] \end{pmatrix}, \frac{1}{n}C\right). \tag{9}$$

where C is the covariance matrix of the two-dimensional distribution $(S, S') = \left(\mathrm{check}_{F,\Phi}(x), \mathrm{check}_{F,\Phi}(x')\right)$, defined by:

$$C := \begin{pmatrix} \mathrm{Var}[S] & \mathrm{Cov}[S, S'] \\ \mathrm{Cov}[S, S'] & \mathrm{Var}[S'] \end{pmatrix}.$$

Computing the Joint Likelihood. Like in Sec. 4.4, after observing $\left(\widehat{\Pr}[F(x) \in \Phi], \widehat{\Pr}[F(x') \in \Phi]\right)$, Eq. (9) induces a likelihood on the probabilities $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$:

$$\begin{pmatrix} \Pr[F(x) \in \Phi] \\ \Pr[F(x') \in \Phi] \end{pmatrix} \sim N\left(\begin{pmatrix} \widehat{\Pr}[F(x) \in \Phi] \\ \widehat{\Pr}[F(x') \in \Phi] \end{pmatrix}, \frac{1}{n}C\right). \tag{10}$$

Estimating the covariance matrix C by the empirical variance and covariance of S and S' (see App. B) gives us an approximate joint likelihood on $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$.

Ratio of Gaussian Random Variables. We use this approximate joint likelihood of $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ to compute an approximate cumulative distribution function (CDF) of $\epsilon(x, x', \Phi)$ based on known work (Thm. 4.4).

The key insight is to view $\epsilon(x, x', \Phi)$ as the logarithm of a ratio of Gaussian random variables, which allows us to apply Thm. 4.4, yielding an approximate CDF for $\epsilon(x, x', \Phi)$, which in turn allows us to compute the approximate probability that $\epsilon(x, x', \Phi)$ lies in the interval $[\hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon, \hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon]$, for some Δ_ϵ.

Theorem 4.4 (Ratio of normal distributions [19]). *Let (X, X') be two random variables that are normally distributed according to:*

$$\begin{pmatrix} X \\ X' \end{pmatrix} \sim N\left(\begin{pmatrix} \mu \\ \mu' \end{pmatrix}, \begin{pmatrix} \sigma^2 & \rho\sigma\sigma' \\ \rho\sigma\sigma' & (\sigma')^2 \end{pmatrix}\right).$$

where σ and σ' are the standard deviations of X and X' respectively, and ρ is the correlation of X and X'.

Then, the cumulative distribution function (CDF) of the ratio X/X' is given by:

$$\Pr\left[\frac{X}{X'} \leq w\right] = L\left(\frac{\mu - \mu' w}{\sigma\sigma' a(w)}, -\frac{\mu'}{\sigma'}; \frac{\sigma' w - \rho\sigma}{\sigma\sigma' a(w)}\right) + L\left(\frac{\mu' w - \mu}{\sigma\sigma' a(w)}, \frac{\mu'}{\sigma'}; \frac{\sigma' w - \rho\sigma}{\sigma\sigma' a(w)}\right).$$

where $a(w) = \sqrt{\frac{w^2}{\sigma^2} - \frac{2\rho w}{\sigma\sigma'} + \frac{1}{(\sigma')^2}}$ and $L(h, k; \gamma)$ is the standard bivariate normal integral (see App. C).

We apply Thm. 4.4 for $X = \Pr[F(x) \in \Phi]$ and $X' = \Pr[F(x') \in \Phi]$. Instantiating the parameters of Thm. 4.4 according to Eq. (10) yields $\mu = \widehat{\Pr}[F(x) \in \Phi]$, $\mu' = \widehat{\Pr}[F(x') \in \Phi]$ and $\begin{pmatrix} \sigma^2 & \rho\sigma\sigma' \\ \rho\sigma\sigma' & (\sigma')^2 \end{pmatrix} = \frac{1}{n}C$.

To compute an approximate CDF on $\epsilon(x, x', \Phi)$, we set w to $\exp(u)$, which yields

$$\Pr\Bigg[\underbrace{\log \frac{\Pr[F(x) \in \Phi]}{\Pr[F(x') \in \Phi]}}_{\epsilon(x, x', \Phi)} \leq u\Bigg] = \Pr\left[\frac{\Pr[F(x) \in \Phi]}{\Pr[F(x') \in \Phi]} \leq \underbrace{\exp(u)}_{w}\right].$$

From this approximate CDF of $\epsilon(x, x', \Phi)$, we want to determine an approximate interval for $\epsilon(x, x', \Phi)$ with confidence $1 - \alpha$. Formally, this means we want to select the smallest Δ_ϵ such that

$$\underbrace{\Pr\left[\hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon \leq \epsilon(x, x', \Phi) \leq \hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon\right]}_{=\Pr[\epsilon(x, x', \Phi) \leq \hat{\epsilon}(x, x', \Phi) + \Delta_\epsilon] - \Pr[\epsilon(x, x', \Phi) \leq \hat{\epsilon}(x, x', \Phi) - \Delta_\epsilon]} \geq 1 - \alpha.$$

Because finding an analytic solution for this equation is hard, we apply binary search to find the smallest Δ_ϵ that satisfies it.

Example. We show how to use the improved approach to estimate $\epsilon(x, x', \Phi)$ on some (x, x', Φ). Assume that we have obtained $n = 10^7$ samples $\mathrm{check}^1_{F,\Phi}(x), \dots, \mathrm{check}^n_{F,\Phi}(x)$ with mean $\frac{1}{n}\sum_{i=1}^{n} \mathrm{check}^i_{F,\Phi}(x) \approx 3.24\%$ and empirical variance $\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x)\right] \approx 3.13\%$ (computed according to App. B). Analogously, we have also obtained n samples with mean $\frac{1}{n}\sum_{i=1}^{n} \mathrm{check}^i_{F,\Phi}(x') \approx 3.04\%$ and empirical variance $\widehat{\mathrm{Var}}\left[\mathrm{check}_{F,\Phi}(x')\right] \approx 2.95\%$. In addition, the empirical correlation between $\mathrm{check}^i_{F,\Phi}(x)$ and $\mathrm{check}^i_{F,\Phi}(x')$ is $\rho = 0.97$. Based on the samples, we compute $\hat{\epsilon}(x, x', \Phi) \approx \log \frac{3.24\%}{3.04\%} \approx 6.4\%$.

We then derive a joint likelihood for $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ according to Eq. (10):

$$N\left(\begin{pmatrix} 3.04\% \\ 3.24\% \end{pmatrix}, \frac{1}{n}\begin{pmatrix} (3.13\%)^2 & 0.97 \cdot 3.13\% \cdot 2.95\% \\ 0.97 \cdot 3.13\% \cdot 2.95\% & (2.95\%)^2 \end{pmatrix}\right).$$

Using Thm. 4.4 with lower bound $l = 6.4\% - \Delta_\epsilon$ and upper bound $u = 6.4\% + \Delta_\epsilon$ for $\Delta_\epsilon = 0.02\%$ allows us to derive that:

$$\Pr\left[6.4\% - \Delta_\epsilon \leq \epsilon(x, x', \Phi) \leq 6.4\% + \Delta_\epsilon\right] =$$
$$\Pr\left[\epsilon(x, x', \Phi) \leq 6.4\% + \Delta_\epsilon\right] - \Pr\left[\epsilon(x, x', \Phi) \leq 6.4\% - \Delta_\epsilon\right] \geq 99\%$$

By adapting Δ_ϵ, we can search for any desired confidence $1 - \alpha$, e.g., for $\alpha = 0.1\%$.

Discussion. Recall that in Sec. 4.4, we first estimated confidence intervals for $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ separately to then combine them using the union bound. In contract, we now first combine the joint distribution of $\Pr[F(x) \in \Phi]$ and $\Pr[F(x') \in \Phi]$ to a distribution of $\epsilon(x, x', \Phi)$, and only then derive a confidence interval. Even if $\rho = 0$ (i.e., without correlation), the approach in this section yields slightly better results, because it takes into account that it is unlikely that both $\widehat{\Pr}[F(x) \in \Phi]$ and $\widehat{\Pr}[F(x') \in \Phi]$ are inaccurate estimates simultaneously.

Correlating random choices is a known technique, see e.g., [23]. However, this technique is usually applied for the difference of random variables, while we apply it for their ratio. In addition, this technique is particularly suitable for applications in the context of algorithms that make random choices.

5 A SEARCH FOR LARGE VIOLATIONS

In the previous section, for a given triple (x, x', Φ), we showed how to replace $\epsilon(x, x', \Phi)$ by an estimate $\hat{\epsilon}(x, x', \Phi)$, where our goal was to construct it with as few samples as possible, while still estimating a tight confidence interval, with high probability. In this section, we address the challenge of finding inputs that induce a large privacy violation. To this end, show how to transform $\hat{\epsilon}(x, x', \Phi)$ to a *differentiable* function $\hat{\epsilon}^d(x, x', \Phi)$. Using $\hat{\epsilon}^d(x, x', \Phi)$, we can define a surrogate optimization problem

$$\begin{array}{ll} \arg\max_{x, x', \Phi} & \hat{\epsilon}^d(x, x', \Phi), \\ \text{s.t. } (x, x') \in \textbf{Neigh} & \end{array}$$

which is differentiable, and can thus be solved with off-the-shelf numerical optimizers.

We begin this section by explaining how to transform $\hat{\epsilon}(x, x', \Phi)$ to $\hat{\epsilon}^d(x, x', \Phi)$ (Sec. 5.1) and then present the surrogate optimization problem and the search for violations (Sec. 5.2).

5.1 From $\hat{\epsilon}(x, x', \Phi)$ to $\hat{\epsilon}^d(x, x', \Phi)$

To obtain $\hat{\epsilon}^d(x, x', \Phi)$ from $\hat{\epsilon}(x, x', \Phi)$, we merely need to translate the (deterministic) programs $\text{check}^i_{F,\Phi}$ to differentiable programs. To understand why, recall the definition of $\hat{\epsilon}(x, x', \Phi)$:

$$\hat{\epsilon}(x, x', \Phi) = \log \frac{\frac{1}{n}\sum_{i=1}^{n} \texttt{check}^i_{F,\Phi}(x)}{\frac{1}{n}\sum_{i=1}^{n} \texttt{check}^i_{F,\Phi}(x')}. \tag{11}$$

In $\hat{\epsilon}(x, x', \Phi)$, n is a constant, and the $\texttt{check}^i_{F,\Phi}(x')$ programs have constants instead of each random choice (e.g., as in Fig. 4). Thus, the sources of non-differentiability are restricted to the statements in $\text{check}^i_{F,\Phi}$. While translating arbitrary statements to differentiable functions is not trivial, we identify a class of programs, for which the translation (1) captures nicely the statements' semantics and (2) can be done systematically. Given the translation, we transform each $\texttt{check}^i_{F,\Phi}(x)$ to a differentiable program $\texttt{dcheck}^i_{F,\Phi}(x)$, which results in a differentiable estimate of $\hat{\epsilon}(x, x', \Phi)$, given by:

$$\hat{\epsilon}^d(x, x', \Phi) = \log \frac{\frac{1}{n}\sum_{i=1}^{n} \texttt{dcheck}^i_{F,\Phi}(x)}{\frac{1}{n}\sum_{i=1}^{n} \texttt{dcheck}^i_{F,\Phi}(x)} \approx \hat{\epsilon}(x, x', \Phi).$$

We note that because the translation from $\texttt{check}^i_{F,\Phi}(x)$ to the differentiable $\texttt{dcheck}^i_{F,\Phi}(x)$ does not preserve semantics, while

$$\begin{array}{rcl}
\neg B & \rightsquigarrow & 1 - B \\
E_1 == E_2 & \rightsquigarrow & e^{-c^2 \cdot (E_1 - E_2)^2} \\
E_1 \leq E_2 & \rightsquigarrow & \left(1 + e^{-c_2 \cdot (E_2 - E_1)}\right)^{-1} \\
B_1 \mathbin{\&\&} B_2 & \rightsquigarrow & B_1 \cdot B_2 \\
B_1 \mathbin{||} B_2 & \rightsquigarrow & B_1 + B_2 - B_1 \cdot B_2 \\
\textbf{if } (B):\ \{x = E_1\} & \rightsquigarrow & x = B \cdot E_1 + (1 - B) \cdot x \\
\textbf{if } (B):\ \{x = E_1\}\ \textbf{else:}\ \{x = E_2\} & \rightsquigarrow & x = B \cdot E_1 + (1 - B) \cdot E_2
\end{array}$$

Figure 8: Transformation rules to make programs differentiable: v is a constant value, x a variable, B, B_1, B_2 Boolean expressions, and E_1, E_2 differentiable arithmetic expressions.

$\hat{\epsilon}^d(x, x', \Phi)$ is approximately equal to $\hat{\epsilon}(x, x', \Phi)$ in practice, the two are not the same in general.

We next describe the class of programs that `DP-Finder` can translate to differentiable programs, and then describe the translation. We note that `DP-Finder` can also handle programs which are not part of this class by replacing the search through optimization with random sampling. However, as we show in Sec. 6, this results in triples with lower privacy violation.

Supported Programs. We focus on a class of programs in which the sources of non-differentiability are conditional statements and Boolean expressions. Concretely, we focus on programs consisting of variables, constants, assignments, differentiable arithmetic expressions (e.g., `x+4`), Boolean expressions (e.g., `x+4≥0 || x<1`) and conditional statements whose branches consist of a single assignment statement. Although loops are not supported in general, if their number of iterations is known at compile-time, they can be unrolled, resulting in a sequential composition of statements (which is supported).

Making Programs Differentiable. Given a $\text{check}^i_{F,\Phi}$ in the aforementioned class, we translate it to a differentiable program. To this end, we define rules, which are applied to each statement separately. The idea is to transform the conditions to functions which have a value close to 1 if their arguments satisfy the original conditions, or a value close to 0 otherwise.

Transformation Rules. Fig. 8 shows our transformation rules for operations that need to be translated. Constants, variables, assignments, and sequential composition remain the same. A negation $\neg B$ is transformed to $1 - B$, and B is then recursively transformed to a differentiable function with our rules. Equality comparison of two arithmetic expressions $E_1 == E_2$ is transformed to an exponential function in $E_1 - E_2$, which is close to 1 if $E_1 == E_2$, and rapidly drops to 0 otherwise. The rule is parametrized by a large constant c (e.g., $c = 50$), which controls how close the transformed program is to the original expression: $c = \infty$ yields the semantics of $E_1 == E_2$. Inequality comparison of two arithmetic expressions $E_1 \leq E_2$ is transformed to a sigmoid function on the (scaled) expression $E_2 - E_1$, which is 1 if $E_1 \ll E_2$, and 0 if $E_1 \gg E_2$. This rule is parametrized by two large constants c_1 and c_2, which have the same effect as c in the previous rule. A logical and expression $B_1 \mathbin{\&\&} B_2$ is transformed to $B_1 \cdot B_2$, which is close to 0 if either expression is close to 0, and close to 1 if both expressions are close to 1. The conditions B_1

and B_2 are then recursively transformed to differentiable functions. Similarly, logical or $B_1 \mid\mid B_2$ could be transformed using the same transformation as for $\neg(\neg B_1 \&\& \neg B_2) \rightsquigarrow 1 - ((1 - B_1) \cdot (1 - B_2))$, but we rewrite it to the (slightly more compact) $B_1 + B_2 - B_1 \cdot B_2$. The if-else statement is transformed to a linear combination of both branches based on the condition B. If there is no else branch, we treat the (missing) else branch as if it were present and contained the assignment $x = x$.

Example. Fig. 9 shows the $\texttt{check}^1_{\texttt{AT},\{[0,0]\}}$ from Fig. 4 and its transformed, differentiable program. For clarity, we extract the conditions of the if-else statements into separate variables, B_1 and B_2. The transformation uses the rules for $==$, $\leq$, $\mid\mid$, and an if-else statement. We assume $c = 50$ for the rules transforming $==$ and $\leq$.

5.2 Differentiable Optimization

Having defined $\hat{\epsilon}^d(x, x', \Phi)$, we can now phrase a surrogate optimization problem for the optimization problem defined in (3):

$$\begin{array}{ll} \arg\max_{x,x'} & \hat{\epsilon}^d(x, x', \Phi) \\ \text{s.t. } (x, x') \in \textbf{Neigh} & \end{array} \tag{12}$$

Since the objective is differentiable (except for a few edge cases, which we shortly discuss), this problem can be solved with gradient methods. In particular, `DP-Finder` uses the Sequential Least Squares Programming (SLSQP) optimizer that also allows to express the constraint $(x, x') \in \textbf{Neigh}$ as-is. We note that the objective function, $\hat{\epsilon}^d(x, x', \Phi)$, is not necessarily convex (consider e.g., the statement $y = x \cdot x \cdot x$), and thus gradient methods may not converge to a global maximum. Nevertheless, similar to many common problems (e.g., training machine learning models), gradient methods may still converge to values close to the optimum. We next discuss edge cases, in which we do not optimize, and the sources of imprecision that arise when considering the surrogate optimization problem (instead of the original one).

Edge Cases. Due to the structure of $\hat{\epsilon}^d(x, x', \Phi)$, defined by $\log(f_1/f_2)$ for differentiable functions $f_1 \approx \Pr[F(x) \in \Phi]$ and $f_2 \approx \Pr[F(x') \in \Phi]$, if the denominator f_2 is zero, the function $\hat{\epsilon}^d(x, x', \Phi)$ is not defined, and thus cannot be optimized. Thus, before running the optimizer on $\hat{\epsilon}^d(x, x', \Phi)$, `DP-Finder` checks if this is the case. This can happen if the probabilities that are sampled are too small, and for the specific samples picked, the condition $[\cdot \in \Phi]$ is never satisfied.

If only the denominator f_2 is 0 and the nominator f_1 is not, then by the definition of DP, we have

$$0 < \underbrace{\Pr[F(x) \in \Phi]}_{\approx f_1} \leq \exp(\epsilon) \underbrace{\Pr[F(x') \in \Phi]}_{\approx f_2} = 0,$$

which implies that DP does not hold for any ϵ. This situation is sometimes referred to as ∞-DP. In this case, there is actually no need to run optimization, and `DP-Finder` reports the current triple (x, x', Φ) as the optimal triple.

If both nominator and denominator are 0, then by the definition of DP, we have $0 = \Pr[F(x) \in \Phi] \leq \exp(\epsilon) \Pr[F(x') \in \Phi] = 0$. Hence, no matter what ϵ we pick, ϵ-DP will never be violated. In this case, `DP-Finder` skips the current triple and continues to the next iteration, where a new triple is randomly picked. In any

```
def check$^1_{AT,\{[0,0]\}}$(x):
  $\rho$ = 7.5
  $\nu$[1] = -23.3
  if x[1]+$\nu$[1] ≥ T+$\rho$:
    y[1] = 1
  else
    y[1] = 0
  $\nu$[2] = 24.3
  if x[2]+$\nu$[2] ≥ T+$\rho$:
    y[2] = 1
  else
    y[2] = 0
  return y[1] == 0 &&
         y[2] == 0
```

```
def dcheck$^1_{AT,\{[0,0]\}}$(x):
  $\rho$ = 7.5
  $\nu$[1] = -23.3
  $B_1 = (1 + e^{-50\cdot(x[1]+\nu[1]-T-\rho)})^{-1}$
  $y[1] = B_1 \cdot 1 + (1 - B_1) \cdot 0$

  $\nu$[2] = 24.3
  $B_2 = (1 + e^{-50\cdot(x[2]+\nu[2]-T-\rho)})^{-1}$
  $y[2] = B_2 \cdot 1 + (1 - B_2) \cdot 0$

  return $e^{-50^2\cdot(y[1]-0)^2}\cdot$
         $e^{-50^2\cdot(y[2]-0)^2}$
```

Figure 9: $\texttt{check}^1_{\texttt{AT},\{[0,0]\}}$ (Fig. 4) and its corresponding differentiable program.

other case, that is, when the denominator is not 0, `DP-Finder` runs optimization.

Sources of Imprecision. The surrogate optimization problem induces two sources of imprecision, which may prevent us from reaching an optimal solution for the original optimization in Eq. (3). The sources of imprecision are: (i) the maximum of $\hat{\epsilon}^d(x, x', \Phi)$ may be different from $\epsilon(x, x', \Phi)$, and (ii) the optimizer may overfit to the random choices fixed in the $\texttt{dcheck}^i_{F,\Phi}(x)$ programs. In Sec. 6, we show empirically that the values for $\hat{\epsilon}^d(x, x', \Phi)$ we find for the transformed programs are close to the true values of $\epsilon(x, x', \Phi)$. Thus, we view this transformation as a heuristic that allows us to apply off-the-shelf optimization techniques to a surrogate problem. To verify the obtained solution, at the end of the execution, `DP-Finder` uses a symbolic solver (PSI [16]), or may estimate the privacy violation with $\hat{\epsilon}(x, x', \Phi)$ if the symbolic solver times out.

6 EVALUATION

We now present a detailed evaluation of our approach.

6.1 Implementation

We implemented a prototype of `DP-Finder` in Python, using the Sequential Least Squares Programming optimizer (SLSQP) from TensorFlow [1] for the optimization task. Our prototype supports algorithms from $\mathbb{R}^n$ to $\mathbb{R}^n$ or to D^n, for a finite set D. Given an algorithm[4], `DP-Finder` randomly picks a triple and draws $n' = 2000$ samples. Then, `DP-Finder` doubles n' until the confidence interval's diameter drops below $4 \cdot 10^{-3}$ (which implies that the error Δ_ϵ is at most $2 \cdot 10^{-3}$). Then, it synthesizes a new counterexample (x, x', Φ) by maximizing the violation $\hat{\epsilon}^d(x, x', \Phi)$, using SLSQP, while satisfying two constraints: (i) x and x' are neighbors, and (ii) $\widehat{\Pr}[F(x) \in \Phi] \geq 10^{-2}$. The second constraint directs the search towards counterexamples whose probability is easier to estimate. If

[4]Our prototype currently does not support an automated synthesis of $\texttt{dcheck}_{F,\Phi}$; instead it assumes to be given $\texttt{dcheck}_{F,\Phi}$ as input (in which case the sampling effort is estimated directly on $\hat{\epsilon}^d(x, x', \Phi)$). We note that implementing this is not a technical challenge, and simply requires to parse the algorithm and apply our rules.

the optimization returns an invalid triple (this can happen, e.g., if $\widehat{\Pr}[F(x) \in \Phi] = 0$, in which case SLSQP fails to enforce constraint (ii)), DP-Finder returns the randomly picked counterexample. Finally, DP-Finder returns the new counterexample (x, x', Φ) and its estimated violation $\hat{\epsilon}^d(x, x', \Phi)$, and continues to the next iteration. In our experiments, we set the number of iterations to 50 (i.e., DP-Finder computes 50 counterexamples for each evaluated algorithm, and finally returns the one with the highest privacy violation).

6.2 Evaluated Algorithms

We evaluated DP-Finder on 9 algorithms from the DP literature, described next (full implementation is given in App. D).

Above Threshold and Variants. We evaluate DP-Finder on AboveThreshold (denoted AT), as defined in Fig. 2, and variants of it. The variants are algorithms 1–5 from [26], which we denote by AT1–AT5. The variations of AboveThreshold are interesting because (i) some of them turned out not to be differentially private and (ii) they obfuscate their input in a non-trivial fashion by adding noise to multiple variables, making them hard to analyze. In the experiments, we fix the size of the input array to 4.

We next describe the variants. Unlike AT, which returns all indices above the threshold, AT1–AT4 only report the first c indices above the threshold, for some meta-parameter c. In our experiments, we set $c = 1$ (we could also use other values of c). Additionally, compared to AT, AT1 uses a different scale for the noise added to the inputs. Compared to AT1, the main difference of AT2 is resampling the threshold noise whenever an input is above the threshold. We note that DP-Finder suggests that this does not increase the privacy of the algorithm, a hypothesis supported by the known upper bounds. Compared to AT1, the main difference of AT3 is that it returns the (noisy) entries that are above the threshold (see Fig. 10a). Hence, AT3 is an algorithm from $\mathbb{R}^k$ to $\mathbb{R}^k$ (while the rest are algorithms to Boolean arrays, i.e., $\{0, 1\}^k$). Compared to AT1, AT4 uses different scales for both the threshold and input noise. Lastly, compared to AT, AT5 does not add noise to the input. This leads to a non-private algorithm (DP-Finder correctly detects this).

For all these algorithms, two array inputs x and x' are neighbors if they differ element-wise by at most one (i.e., $\mathbf{Neigh}_{\leq 1}$): $\forall i \in \{0, \dots, k-1\}. |x_i - x'_i| \leq 1$.

The known upper bounds on the differential privacy of these algorithms are: AT is 0.45-DP, AT1 and AT2 are 0.1-DP, AT3 is 0.2-DP, AT4 is 0.175-DP, and AT5 is ∞-DP. The latter means that there are two neighboring inputs and an output set which can be returned for one of the inputs but not for the other. In all our result graphs, we show these upper bounds in a blue line (to put in context the lower bound results).

To select Φ, in all algorithms except AT3, DP-Finder samples a single output y, uniformly at random, from all possible outputs of these algorithms, and sets $\Phi := \{y\}$. For AT3, this is not possible, because its output is continuous, and thus $\Pr[F(x) \in \{y\}] = 0$, for any y. Instead, DP-Finder picks Φ to be the box $\Phi := \{y \in \mathbb{R}^n \mid \forall i \in \{0, \dots, k-1\}. a_i \leq y_i \leq b_i\}$. To sample (a_i, b_i), it first picks an array of indicators I_i, such that $I_i = 1$ indicates that the i^{th} value lies above the threshold, uniformly at random from all possibilities. Then, it sets $(a_i, b_i) = (-10 - 3, -10 + 3)$

```
def AT3(x):
  ρ = Lap(20)
  for i = 1 to k:
    ν[i] = Lap(20)
    if x[i]+ν[i]≥T+ρ:
      y[i] = x[i]+ν[i]
    else
      y[i] = -10
  return y
```

(a) Variant of AT

```
def noisyMax(x):
  best = 0
  r = 0
  for i = 1 to k:
    d = x[i]+Lap(20)
    if d>best or i==0:
      r = i
      best = d
  return r
```

(b) NoisyMax

Figure 10: Two representative algorithms used for evaluation.

if $I_i = 0$ (note that -10 is the value returned for entries which are not above the threshold), and $(a_i, b_i) = (x_i - 3, x_i + 3)$, otherwise.

Noisy Maximum. We also evaluate on two algorithms taken from [3]. The first is noisyMax (Fig. 10b), which is a noisy implementation of a function returning the index of the largest element in an array. Here, the noise is drawn from a Laplace distribution. The second algorithm is expMech, which is identical to noisyMax but draws the noise from an exponential distribution. Both algorithms are known to be 0.1-DP. Just as for AT, DP-Finder uses the neighboring notion of $\mathbf{Neigh}_{\leq 1}$ and picks $\Phi := \{y\}$, for y picked uniformly at random from all possible outputs.

Sum. To illustrate a different notion of neighboring inputs, we also evaluate on sum [12], which takes an array x, whose entries are between -1 and 1, and returns its noisy sum. Here, two arrays x and x' are neighbors if x' is x extended with an additional entry. In this benchmark, we consider a single $\Phi := \{x \in \mathbb{R} \mid a \leq x \leq b\}$, where $a = \sum_{i=1}^{k} x_i - 3$ and $b = \sum_{i=1}^{k} x_i + 3$.

6.3 Evaluation Results

Our evaluation results answer the following questions:

- **Q1** How precise are the estimated violations $\hat{\epsilon}^d(x, x', \Phi)$, compared to $\epsilon(x, x', \Phi)$?
- **Q2** How efficient is DP-Finder in finding violations compared to random search?
- **Q3** How efficient is DP-Finder in terms of runtime?

We ran all experiments on a machine with 500GB RAM and 128 cores at 1.2GHz, running Ubuntu 16.04.3 LTS with Tensorflow 1.9.0 and Python 3.5.2.

Q1: Precision of estimated violations. To evaluate the precision of the estimated violations, we compare the estimated violation $\hat{\epsilon}^d(x, x', \Phi)$ with the actual violation $\epsilon(x, x', \Phi)$, as computed by the exact solver PSI [16]. Fig. 11 shows the boxplots of the estimated violation $\hat{\epsilon}^d(x, x', \Phi)$ and the actual violation $\epsilon(x, x', \Phi)$, obtained from the 50 counterexamples generated by DP-Finder for each algorithm. The figure shows that our estimation is very precise, expect in a few cases (e.g., for AT5). We recall that the imprecision of $\hat{\epsilon}^d(x, x', \Phi)$ is due to (i) the finite sampling of the randomized programs $\text{check}_{F,\Phi}(x)$ (presented in Sec. 4) and (ii) the transformation of the individual samples $\text{check}^i_{F,\Phi}(x)$ to differential functions

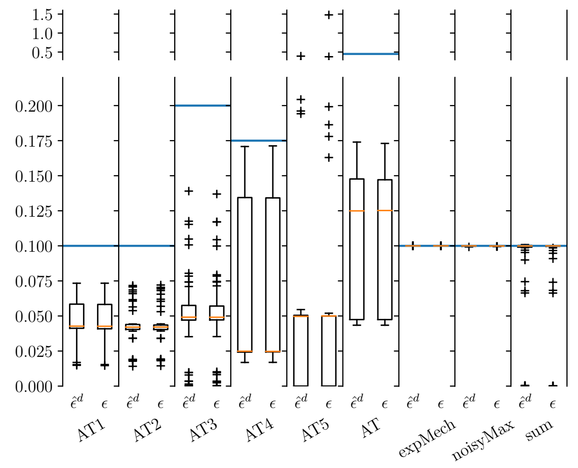


Figure 11: Boxplot comparing the estimated violation $\hat{\epsilon}^d(x, x', \Phi)$ to the true violation $\epsilon(x, x', \Phi)$. The solid blue lines show known upper bounds of these algorithms. We omit one counterexample with a violation of ∞ for AT5.

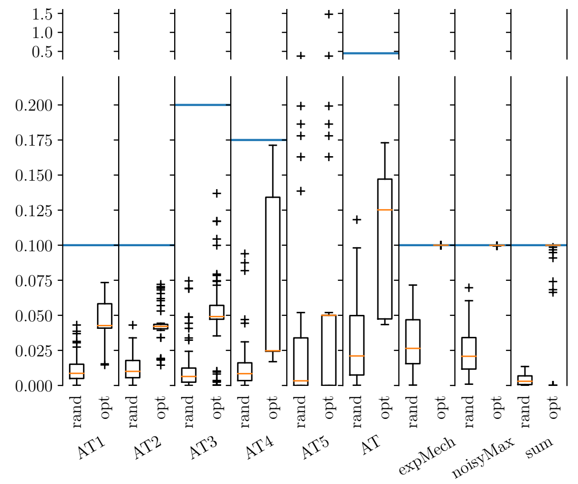


Figure 12: Boxplot of the true violation $\epsilon(x, x', \Phi)$ found by randomly picking a triple vs. our search. The solid blue lines show known upper bounds of these algorithms. We omit one counterexample with a violation of ∞ (for both the randomly picked and the optimized counterexamples) on AT5.

(presented in Sec. 5). In particular, Fig. 11 demonstrates that both steps do not significantly reduce the quality of the estimates. As the differentiable estimate $\hat{\epsilon}^d(x, x', \Phi)$ is more imprecise compared to the estimate $\hat{\epsilon}(x, x', \Phi)$ based on sampling, Fig. 11 also demonstrates the effectiveness of our estimation method (Sec. 4).

Q2: Efficiency of Violation Search. Next, we compare the efficiency of DP-Finder in finding counterexamples with large violations compared to random search (where we randomly sample triples). For each algorithm, we computed the exact violation of the

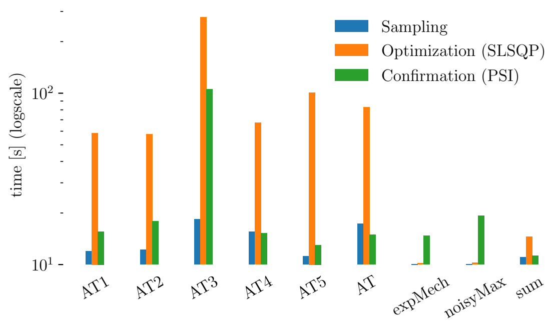


Figure 13: Execution times of DP-Finder for sampling and finding counterexamples using SLSQP.

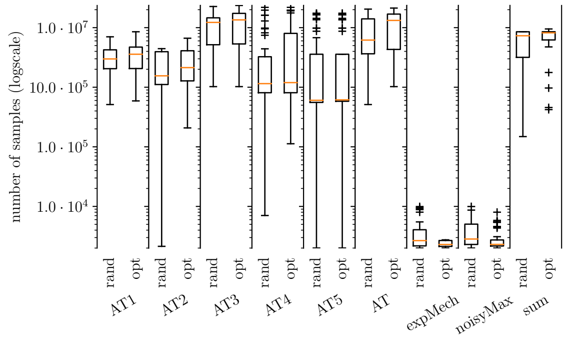


Figure 14: Numbers of samples used by DP-Finder, for random triples (rand) and optimized triples (opt).

50 counterexamples produced by DP-Finder, and compared this to the exact violation of 50 counterexamples obtained by random sampling. Fig. 12 shows the results. Left bars show the privacy violations of the randomly generated counterexamples, while right show those produced by DP-Finder. Results indicate that DP-Finder outperforms the random approach. Comparing the medians of found violations for 50 runs, we get an improvement of a factor of 2 (for AT4) to 33 (for sum), depending on the algorithm. Furthermore, DP-Finder returns counterexamples whose violations are often close to the known upper bounds. In particular, for expMech, DP-Finder found, in all iterations, a counterexample whose violation was very close to the known upper bound. This demonstrates that these are tight bounds. For the other algorithms, it is unclear which of the bounds can be tightened (perhaps both).

Q3: Runtime. Finally, we study the runtime of DP-Finder. Fig. 13 displays DP-Finder's execution times for the 9 algorithms. We split the runtime into the two steps of DP-Finder: sampling and numerical optimization. In addition, we report the time spent on confirming the estimated violations (using PSI).

The results indicate that DP-Finder spends most of its time on the optimization problem, which is inherently hard. For every algorithm, each iteration of DP-Finder completes within 5 minutes on average, demonstrating that DP-Finder is efficient in practice.

Fig. 14 shows the number of samples that DP-Finder selected, according to the process described in Sec. 4.5. It demonstrates that the required number of samples varies greatly across algorithms, and even for a given algorithm. This suggests to adaptively select the number of samples, which is what DP-Finder does (see Sec. 4.5).

7 RELATED WORK

In this section, we discuss the work closely related to ours.

Proving Differential Privacy. DP-Finder computes counterexamples to differential privacy (DP), thereby providing lower bounds to DP. A complementary problem to this is finding upper bounds, thereby proving DP. Many works have studied verification of DP. Some works present languages that, at compile time, determine the privacy or sensitivity of algorithms (queries) [6, 15, 32, 35]. A different approach translates probabilistic algorithms to formulas in Hoare logic to verify privacy [5]. Recently, proofs by couplings have been shown successful for verifying privacy [3, 4].

Proving Sensitivity. Dwork et al. [12] defined the (global) *sensitivity* of a function f as the maximum amount that any input to f can change the output. While determining this sensitivity can be done analytically for some functions (e.g., noisy sum), in others, this task is more complex. In Nissim et al. [28], the authors define the *smooth sensitivity* of a function for a given database, to avoid the pessimistic worst-case bound of sensitivity. They also present a sampling approach to approximate the smooth sensitivity. In Rubinstein and Aldà [33], the authors present a sampling approach to approximate the (global) sensitivity.

Lower Bounds. The study of lower bounds on privacy started with the work of Dinur and Nissim [11]. While they do not define privacy in this work, they show how much noise needs to be added to prevent a gross privacy violation. Since then, differential privacy has been formally defined in [12] as $(\epsilon, 0)$- and (ϵ, δ)-privacy, with which lower bounds were proven for certain algorithms. Hardt and Talwar [18] provide lower bounds on different noise mechanism using ideas from convex geometry. [10] improve some of their lower bounds and study additional settings (e.g., the one of [11]). [25] study lower bounds in the context of how big a database has to be to guarantee privacy.

Making Algorithms Differentiable. Priya Inala et al. [30] synthesize unknowns in an algorithm that involves discrete and floating point computation. To search for the unknowns, they make the algorithm differentiable by techniques similar to ours, e.g, they use the same construction to make the if-then-else primitive differentiable.

8 FUTURE WORK

In this section, we discuss possible future work items.

Extending DP-Finder to Real-world Algorithms. An important topic of future research is extending DP-Finder to real-world algorithms, like [8, 14, 29]. The main gap is that DP-Finder, in its current form, does not scale to such complex algorithms. We see several ways to mitigate this issue: (i) exploiting properties of the search space, e.g., if dense, randomly sampling triples may perform comparably to optimizing them, (ii) employing other optimization algorithms (e.g., MCMC), which may lead to speed-ups, or (iii) decreasing the confidence, which will reduce the number of required samples, and hence result in a smaller term to optimize.

Extending DP-Finder to Noise Depending on Inputs. In its current form, DP-Finder exploits that the noise terms do not depend on the input by (i) using the same noise for both inputs during the sampling and (i) changing only the inputs, but not the noise during the search.

To analyze algorithms whose noise terms depend on the inputs, we can address (i) by changing the sampling method, e.g., to Sec. 4.3 or Sec. 4.4. We note that this will result in a performance decrease.

However, when using the fixed noise for values slightly different from x (in particular during optimization), the noise comes from a slightly different distribution (sampled based on the original x). To compensate this slight error, we can adapt DP-Finder to use importance sampling, i.e., introduce larger weights for randomness which does not get sampled often enough.

Expectation-preserving Program Transformations. We also experimented with expectation-preserving program transformations for $\text{check}_{F,\Phi}$, i.e., we modified the program, resulting in a program which (i) exhibits the same expectation $\mathbb{E}\left[\text{check}_{F,\Phi}(x)\right]$ and (ii) allows for more efficient sampling.

We found that expectation-preserving code transformations can improve the results of DP-Finder, i.e., they can (i) reduce the number of samples required to get a small confidence interval and (ii) improve the quality of the violations found by the search.

However, the expectation-preserving transformations we applied were manual, and often required knowledge about the program under investigation. A principled approach that can detect and apply a large set of expectation-preserving transformations could improve the performance of DP-Finder further.

9 CONCLUSION

We presented a new approach and a corresponding system, called DP-Finder, that finds privacy violations in randomized algorithms. These violations establish lower bounds on the differential privacy enforced by these algorithms. This is useful as it allows one to establish tightness of existing upper bounds or find violations for incorrect upper bounds.

DP-Finder finds large privacy violations by leveraging two technical insights. First, we defined an estimate of the privacy violation through correlated sampling. We use a carefully-designed heuristic to determine the sampling effort necessary to use as few samples as possible, while still estimating a tight confidence interval for the estimated violation. Second, we introduced rewrite rules that transform the estimated (non-differentiable) violation into a differentiable function, which can then be given to numerical optimizers to search for large privacy violations.

We evaluated DP-Finder on a number of randomized algorithms from the DP literature. Our results show that DP-Finder finds large privacy violations, often close to the known upper bounds, demonstrating its practical promise.

ACKNOWLEDGMENTS

The research leading to these results was partially supported by an ERC Starting Grant 680358.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proc. Operating Systems Design and Implementation (OSDI)*. 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proc. Conference on Computer and Communications Security (CCS)*. 308–318. https://doi.org/10.1145/2976749.2978318

[3] Aws Albarghouthi and Justin Hsu. 2017. Synthesizing Coupling Proofs of Differential Privacy. *Proc. ACM Program. Lang.* 2, POPL, Article 58 (2017), 30 pages. https://doi.org/10.1145/3158146

[4] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017. Proving uniformity and independence by self-composition and coupling. In *International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*. 19. https://hal.sorbonne-universite.fr/hal-01541198

[5] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, César Kunz, and Pierre-Yves Strub. 2014. Proving Differential Privacy in Hoare Logic. In *Proc. Computer Security Foundations Symposium (CSF)*. 411–424. https://doi.org/10.1109/CSF.2014.36

[6] Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin Pierce. 2016. Programming Language Techniques for Differential Privacy. *ACM SIGLOG News* 3, 1 (2016), 34–53. https://doi.org/10.1145/2893582.2893591

[7] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proc. Symposium on Operating Systems Principles (SOSP)*. 441–459. https://doi.org/10.1145/3132747.3132769

[8] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. 2005. Practical Privacy: The SuLQ Framework. In *Proc. Principles of Database Systems (PODS)*. 128–138. https://doi.org/10.1145/1065167.1065184

[9] Y. Chen and A. Machanavajjhala. 2015. On the Privacy Properties of Variants on the Sparse Vector Technique. (2015). arXiv:cs.DB/1508.07306

[10] Anindya De. 2012. Lower Bounds in Differential Privacy. In *Proc. Theory of Cryptography Conference (TCC)*. 321–338. https://doi.org/10.1007/978-3-642-28914-9_18

[11] Irit Dinur and Kobbi Nissim. 2003. Revealing Information While Preserving Privacy. In *Proc. Principles of Database Systems (PODS)*. 202–210. https://doi.org/10.1145/773153.773173

[12] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proc. Theory of Cryptography Conference (TCC)*. 265–284. https://doi.org/10.1007/11681878_14

[13] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9 (2014), 211–407. https://doi.org/10.1561/0400000042

[14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proc. Conference on Computer and Communications Security (CCS)*. 1054–1067. https://doi.org/10.1145/2660267.2660348

[15] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. 2013. Linear Dependent Types for Differential Privacy. In *Proc. Principles of Programming Languages (POPL)*. 357–370. https://doi.org/10.1145/2429069.2429113

[16] Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification (CAV)*. Vol. 9779. 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

[17] Anupam Gupta, Aaron Roth, and Jonathan Ullman. 2012. Iterative Constructions and Private Data Release. In *Proc. Theory of Cryptography Conference (TCC)*. 339–356. https://doi.org/10.1007/978-3-642-28914-9_19

[18] Moritz Hardt and Kunal Talwar. 2010. On the Geometry of Differential Privacy. In *Proc. Symposium on Theory of Computing (STOC)*. 705–714. https://doi.org/10.1145/1806689.1806786

[19] D. V. Hinkley. 1969. On the Ratio of Two Correlated Normal Random Variables. *Biometrika* 56, 3 (1969), 635–639. http://www.jstor.org/stable/2334671

[20] Wassily Hoeffding. 1963. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.* 58, 301 (1963), 13–30. https://doi.org/10.1080/01621459.1963.10500830

[21] Z. Ji, Z. C. Lipton, and C. Elkan. 2014. Differential Privacy and Machine Learning: a Survey and Review. (2014). arXiv:1412.7584

[22] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. *Proc. VLDB Endow.* 11, 5 (2018), 526–539. https://doi.org/10.1145/3177732.3177733

[23] H. Kahn and A. W. Marshall. 1953. Methods of Reducing Sample Size in Monte Carlo Computations. *Journal of the Operations Research Society of America* 1, 5 (1953), 263–278. http://www.jstor.org/stable/166789

[24] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2013. Analyzing Graphs with Node Differential Privacy. In *Theory of Cryptography Conference (TCC)*. 457–476. https://doi.org/10.1007/978-3-642-36594-2_26

[25] Assimakis Kattis and Aleksandar Nikolov. 2017. Lower Bounds for Differential Privacy from Gaussian Width. In *Int. Symposium on Computational Geometry (SoCG)*. 45:1–45:16. https://doi.org/10.4230/LIPIcs.SoCG.2017.45

[26] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the Sparse Vector Technique for Differential Privacy. *Proc. VLDB Endow.* 10, 6 (2017), 637–648. https://doi.org/10.14778/3055330.3055331

[27] Frank D. McSherry. 2009. Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis. In *Proc. SIGMOD International Conference on Management of Data*. 19–30. https://doi.org/10.1145/1559845.1559850

[28] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth Sensitivity and Sampling in Private Data Analysis. In *Proc. Symposium on Theory of Computing (STOC)*. 75–84. https://doi.org/10.1145/1250790.1250803

[29] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar. 2016. Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data. (2016). arXiv:stat.ML/1610.05755

[30] J. Priya Inala, S. Gao, S. Kong, and A. Solar-Lezama. 2018. REAS: Combining Numerical Optimization with SAT Solving. (2018). arXiv:cs.PL/1802.04408

[31] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating Data to Sensitivity in Private Data Analysis. *Proc. VLDB Endow.* 7, 8 (2014), 637–648.

[32] Jason Reed and Benjamin C. Pierce. 2010. Distance Makes the Types Grow Stronger: A Calculus for Differential Privacy. In *Proc. International Conference on Functional Programming (ICFP)*. 157–168. https://doi.org/10.1145/1863543.1863568

[33] Benjamin I. P. Rubinstein and Francesco Aldà. 2017. Pain-Free Random Differential Privacy with Sensitivity Sampling. In *Proc. International Conference on Machine Learning, (ICML)*. 2950–2959.

[34] A. W. van der Vaart. 1998. *Asymptotic Statistics*. https://doi.org/10.1017/CBO9780511802256

[35] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. 2017. A Framework for Adaptive Differential Privacy. *Proc. ACM Program. Lang.* 1, ICFP, Article 10 (2017), 29 pages. https://doi.org/10.1145/3110254

A LIKELIHOOD INDUCED BY GAUSSIAN DISTRIBUTION

Let $X \in \mathbb{R}$ be a constant, $Y \in \mathbb{R}$ be a random variable, and $\sigma^2 \in \mathbb{R}$ be a constant. We write $Y = N\left(X, \sigma^2\right)$ to indicate that Y is sampled from a Gaussian distribution with mean X and variance σ^2.

Then, observing Y induces a likelihood on X, where X is distributed according to a Gaussian distribution. Formally, this means that we can switch X and Y, resulting in $X = N\left(Y, \sigma^2\right)$. We can derivate this as follows:

$$\begin{aligned}
& Y = N\left(X, \sigma^2\right) && \\
\iff\ & Y = X + N\left(0, \sigma^2\right) && \text{separate out mean} \\
\iff\ & X = Y - N\left(0, \sigma^2\right) && \text{subtract } N\left(0, \sigma^2\right) \\
\iff\ & X = Y + N\left(0, \sigma^2\right) && \text{Gaussian distribution is symmetric around 0} \\
\iff\ & X = N\left(Y, \sigma^2\right) && \text{combine with mean}
\end{aligned}$$

B SAMPLE VARIANCE AND COVARIANCE

For a random variable S over $\mathbb{R}$ and independent samples $S_1, \ldots, S_n$ from S, the *sample variance* $\widehat{\text{Var}}[S]$ estimates the variance of S:

$$\widehat{\text{Var}}[S] := \frac{\sum_{i=1}^n S_i^2 - (\sum_{i=1}^n S_i)^2/n}{n-1} \approx \text{Var}[S]$$

Likewise, for two jointly-distributed random variables S and S' over $\mathbb{R}$ and independent samples $(S_1, S_1'), \ldots, (S_n, S_n')$ from (S, S'), the *sample covariance* $\widehat{\text{Cov}}[S, S']$ estimates the covariance of S and S':

$$\widehat{\text{Cov}}[S, S'] := \frac{\sum_{i=1}^n S_i S_i' - (\sum_{i=1}^n S_i)(\sum_{i=1}^n S_i')/n}{n-1} \approx \text{Cov}[S, S']$$

For large n, the sample variance and covariance are close to the true variance and covariance. In this work, we use $n \geq 10^3$ and work with $\widehat{\text{Var}}[S]$ instead of the true variance $\text{Var}[S]$ and likewise for $\widehat{\text{Cov}}[S, S']$.

C BIVARIATE NORMAL INTEGRAL

Let Y_1 and Y_2 be variables drawn from a normal distribution with correlation coefficient γ:

$$\begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \gamma \\ \gamma & 1 \end{pmatrix}\right)$$

Then, $L(h, k, \gamma)$ is the standard bivariate normal integral computing $\Pr[Y_1 \leq h, Y_2 \leq k]$:

$$L(h, k, \gamma) = \frac{1}{2\pi\sqrt{1-\gamma^2}} \int_h^\infty \int_k^\infty \exp\left(-\frac{x^2 - 2\gamma xy + y^2}{2(1-\gamma^2)}\right) dx dy$$

D CODE

We provide implementations[5] of $\texttt{check}_{F,\Phi}$ for all algorithms F from Sec. 6. The implementations are in PSI, and have a placeholder for the input x (denoted by `[$A]`) and for the check Φ (denoted by `[$O]`). Some algorithms have an additional meta-parameter, denoted by `$C`.

If $F(x) \notin \Phi$, $\texttt{check}_{F,\Phi}$ should return 0, but instead our implementation throws an assertion failure (this is slightly easier to encode in PSI). If $F(x) \in \Phi$, the implementation returns 1, as expected.

In addition, the implementations conflate the computation of the output $F(x)$ with the check $F(x) \in \Phi$, which allows slightly more efficient analysis with PSI.

[5]https://github.com/eth-sri/dp-finder/tree/initial-release/dpfinder/algorithms/psi_implementations