Approximate Algorithms for Verifying Differential Privacy with Gaussian Distributions

Bishnu Bhusal
University of Missouri
Columbia, USA
bhusalb@missouri.edu

Rohit Chadha
University of Missouri
Columbia, USA
chadhar@missouri.edu

A. Prasad Sistla
University of Illinois at Chicago
Discovery Partners Institute
Chicago, USA
sistla@uic.edu

Mahesh Viswanathan
University of Illinois at Urbana-Champaign
Urbana, USA
vmahesh@illinois.edu

Abstract

The verification of differential privacy algorithms that employ Gaussian distributions is little understood. This paper tackles the challenge of verifying such programs by introducing a novel approach to approximating probability distributions of loop-free programs that sample from both discrete and continuous distributions with computable probability density functions, including Gaussian and Laplace. We establish that verifying (ϵ, δ)-differential privacy for these programs is *almost decidable*, meaning the problem is decidable for all values of δ except those in a finite set. Our verification algorithm is based on computing probabilities to any desired precision by combining integral approximations, and tail probability bounds. The proposed methods are implemented in the tool, DiPApprox, using the FLINT library for high-precision integral computations, and incorporate optimizations to enhance scalability. We validate DiPApprox on fundamental privacy-preserving algorithms, such as Gaussian variants of the Sparse Vector Technique and Noisy Max, demonstrating its effectiveness in both confirming privacy guarantees and detecting violations.

CCS Concepts

• **Security and privacy → Logic and verification**; • **Theory of computation → Program analysis**.

Keywords

Differential Privacy, Verification, Gaussian Distribution, Tail Bounds

ACM Reference Format:
Bishnu Bhusal, Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2018. Approximate Algorithms for Verifying Differential Privacy with Gaussian Distributions. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 22 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Differential Privacy [30] has become the standard for protecting individual data in sensitive datasets. In this framework [33], data is managed by a trusted curator and queried by a potentially dishonest analyst. The goal is to ensure that query results remain nearly "unchanged" whether or not an individual's data is included, thereby preserving privacy even against adversaries with unlimited computational power and auxiliary data. However, it is challenging to design protocols that meet this high bar because reasoning about differential privacy is subtle, complex, and involves precise, quantitative reasoning. Many proposed algorithms and proofs have been found flawed [21, 37, 38, 41], leading to growing interest in programming languages, type systems, and formal verification tools for privacy analysis [1, 4–6, 8–12, 18, 18, 19, 35, 36, 40, 43–45, 47–49, 54, 55].

Despite advances, automated verification of differential privacy still faces significant challenges. Differential privacy is often ensured by adding noise, commonly from Laplace and Gaussian (or Normal) distributions. While there exist several automated tools for verifying differential privacy with Laplace distributions [4, 16, 18, 19, 29, 53], we are not aware of any automated tool that can effectively analyze programs that incorporate Gaussian noise.

This paper studies the problem of automatically verifying if a program using Gaussian distributions meets the requirement of differential privacy. While the problem of checking differential privacy is undecidable even for programs that only toss fair coins [4], a decidable subclass of programs has been identified [4]. However, programs in this subclass are not allowed to sample from Gaussian distributions; they have inputs and outputs over finite domains, and can sample from Laplace distributions. The algorithm in [4] requires handling symbolic expressions with integral functionals, and does not generalize to Gaussian distributions as the Gaussian distribution lacks closed-form integral-free expressions for the cumulative distribution. This paper presents an approach to reason about programs that can also sample from the Gaussian distributions. We do assume inputs and outputs are over finite domains, as in [4].

Before presenting our contributions, let us recall the basic setup of differential privacy. The differential privacy program/mechansim is usually parameterized by ϵ, henceforth referred to as *privacy parameter* which controls the noise added during the computation.

There are two additional quantities that influence the protocol and its correctness: a *privacy budget* $\epsilon_{\text{prv}} > 0$, and *error parameter* $\delta \in [0, 1]$, which accounts for the probability of privacy loss.[1] Given a binary relation Φ on inputs called an *adjacency relation*, and taking $\text{Prob}[\epsilon, \text{P}(u) \in F]$ to be the probability that P outputs $o \in F$ on input u, P is said to be $(\epsilon_{\text{prv}}, \delta)$-*differentially private* if for each pair $(u, u') \in \Phi$, and measurable subset of outputs F, we have that

$$\text{Prob}[\epsilon, \text{P}(u) \in F] - e^{\epsilon_{\text{prv}}} \text{Prob}[\epsilon, \text{P}(u') \in F] \leq \delta. \tag{1}$$

When $\delta = 0$, $(\epsilon_{\text{prv}}, 0)$-differential privacy is referred to as pure ϵ_{prv}-differential privacy. We shall say that P is $(\epsilon_{\text{prv}}, \delta)$-*strictly differentially private* if the inequality in (1) holds *strictly* (i.e, we require $<$ instead of $\leq$).

Contributions. Given fixed $\epsilon > 0$, let $\text{Prob}[\epsilon, u, o, \text{P}] = \text{Prob}[\epsilon, \text{P}(u) \in \{o\}]$ denote the probability of P returning a single output o on input u. Let $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$ be a function that returns an interval $[L, U]$ such that $\text{Prob}[\epsilon, u, o, \text{P}] \in [L, U]$ and $|U - L| \leq 2^{-\varrho}$; in other words, $\text{ComputeProb}_{\varrho}(\cdot)$ approximates $\text{Prob}[\epsilon, u, o, \text{P}]$ with desired precision ϱ. Our first result establishes that if the function $\text{ComputeProb}_{\varrho}(\cdot)$ is computable for a class of programs, then the problem of determining if program P in this class is non-$(\epsilon_{\text{prv}}, \delta)$-differential privacy is recursively enumerable (See Theorem 9 on Page 9). Furthermore, we show that the computability of $\text{ComputeProb}_{\varrho}(\cdot)$ also implies that checking if a P is $(\epsilon_{\text{prv}}, \delta)$-strict differentially private, is recursively enumerable (See Theorem 9 on Page 9). These two observations suggest that verifying $(\epsilon_{\text{prv}}, \delta)$-differential privacy is *almost decidable* — the problem of checking if P is $(\epsilon_{\text{prv}}, \delta)$-differentially private is *decidable* for all values of δ, *except* those in a finite set $\text{D}_{\text{P},\epsilon,\epsilon_{\text{prv}},\Phi}$ that depends on the program P, ϵ, ϵ_{prv} and Φ. Informally, $\text{D}_{\text{P},\epsilon,\epsilon_{\text{prv}},\Phi}$ is the set of δ for which the inequality (1) is an equality; the precise definition is given in Definition 4 on Page 7.

Next, we observe that the function $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$ is computable for a large class of programs that sample from distributions such as the Gaussian distribution. Our class, called DiPGauss, consist of *loop-free* programs, with finite domain inputs and outputs. These programs can sample from continuous distributions, provided the distributions have *finite* means and *finite* variances [2] and *computable* [3] probability density functions. This includes commonly used distributions like Gaussian and Laplace. This class of programs differs from the class presented in [4] — while they are loop-free, they allow sampling from Gaussian distributions. Despite their simple structure, DiPGauss programs include many widely used algorithms such as the Sparse Vector Technique (SVT) [31], Noisy Max [29] and their variants with Gaussian mechanisms [56]. These algorithms are used in applications like ensuring the privacy of Large Language Models (LLMs) [2].

To describe our algorithm for $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$, observe that $\text{Prob}[\epsilon, u, o, \text{P}]$ can be written as a sum of the probabilities of *execution paths* of P on input u leading to output o. Since P is loop-free, P has only a finite number of execution paths on each input u. The probability of each execution path can be written as sum of iterated integrals. If none of the integrals have ∞ or $-\infty$ as upper or lower limits, given the assumption that the probability density functions are computable, these integrals can be evaluated to any desired precision.

We can avoid having ∞ or $-\infty$ as limits of integrals by appealing to tail bounds derived from Chernoff or Chebyshev inequalities as follows. Given a threshold $\text{th} > 0$, we can write the probability of an execution path τ as $\text{bpr}(\epsilon, \tau, \text{th}) + \text{tpr}(\epsilon, \tau, \text{th})$. Here $\text{bpr}(\cdot)$ is the probability of the execution τ under the constraint that all sampled values $X_{\mathbf{r}_1}, \ldots, X_{\mathbf{r}_n}$ in the execution remain within $\text{th} \cdot \sigma_i$ from μ_i where μ_i and σ_i are the mean and standard deviation of the distribution of $X_{\mathbf{r}_i}$, respectively. The term $\text{tpr}(\cdot)$ accounts for the tail probability, capturing cases where at least one sampled value deviates by at least $(\text{th} \cdot \sigma)$ from μ. Now, $\text{bpr}(\epsilon, \tau, \text{th})$ can again be written as a sum of iterated integrals with finite limits. The assumption of computability ensures that $\text{bpr}(\cdot)$ can be computed to any desired precision. Moreover, by Chebyschev's inequality, we can always select th such that $\text{tpr}(\epsilon, u, o, \text{th})$ is arbitrarily small. This guarantees that $\text{Prob}[\epsilon, u, o, \text{P}]$ can be computed to any required degree of precision.

Using our algorithm for $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$ and algorithm for almost deciding $(\epsilon_{\text{prv}}, \delta)$-differential privacy, we implement an algorithm $\text{VerifyDP}_{\varrho}$ that returns one of 3 outputs: DP, Not_DP, or Unknown. Crucially, as the precision parameter ϱ decreases, the likelihood of getting Unknown diminishes. Our implementation of $\text{VerifyDP}_{\varrho}$ incorporates several optimizations to improve scalability for practical examples.

First, differential privacy requires verifying that inequality (1) holds for *all* pairs of adjacent inputs u, u' and *all* subsets F of outputs. A key insight is that this verification can be reduced to checking a modified equation (See Lemma 2 on Page 7) where F is taken as the set of all outputs, leading to *exponential* reduction in complexity. Intuitively, it is enough to check the inequality (1) for the set F of all outputs o for which $\text{Prob}[\epsilon, \text{P}(u) = o] - e^{\epsilon_{\text{prv}}} \text{Prob}[\epsilon, \text{P}(u') = o] > 0$.

Next, we observe that the integral nesting depth significantly impacts performance while computing the value of $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$. Thus, we introduce a heuristic to reduce it. Intuitively, each variable being integrated in an integral I during $\text{ComputeProb}_{\varrho}(\epsilon, u, o, \text{P})$ represents a value sampled from an independent distribution in the program. We call such variables, independent random variables. For each nested integral I, we can define a directed acyclic graph, called the *dependency graph* of I, on the independent random variables. An edge from $\mathbf{r}$ to $\mathbf{r}'$ in the dependency graph indicates that a) the integration over variable $\mathbf{r}'$ is nested within integration over $\mathbf{r}$ in I, and that b) $\mathbf{r}$ appears in either the upper or lower limit in the integration over $\mathbf{r}'$.

We observe that if the sets of variables reachable from two variables $\mathbf{r}_1$ and $\mathbf{r}_2$ in the dependency graph of I are disjoint, the nested depth of I can be reduced by separating the integrations over $\mathbf{r}_1$ and $\mathbf{r}_2$. Exploiting this observation, we give an algorithm based on the topological sorting of the dependency graph which yields an integral expression I' that has the same value as I, but lower nesting depth. Applying this heuristic, our prototype tool was able to rewrite the integrals for the SVT-Gauss algorithm [56] with at most 3 nested integrals, independent of the number of input variables.

[1] Often, ϵ_{prv} is expressed as a function of δ, ϵ. A popular choice of ϵ_{prv} is ϵ, which often occurs in case of pure differential privacy.

[2] A continuous distribution may not have finite mean or finite variance.

[3] By computable, we mean computable as defined in recursive real analysis [39].

Without this optimization, the nested depth scales with the number of input variables.

We implemented VerifyDP$_\varrho$ in a tool, DiPApprox. Our implementation leverages the FLINT library [50] to evaluate nested integrals with finite limits. FLINT enables rigorous arithmetic with arbitrary precision using ball enclosures, where values are represented by a midpoint and a radius. Using the above outlined approach, our tool successfully verified variants of the Sparse Vector Technique [31, 56], Noisy Max [33], k-Min-Max [19], m-Range [19] with Gaussians (and Laplacians) across varying input lengths. Additionally, it confirmed the non-differential privacy of insecure versions of the Sparse Vector Technique with Gaussians. These experiments highlight the potential of our approach for the automated verification of differential privacy in programs that sample from complex continuous distributions. DiPApprox is available for download at [3].

Organization. The rest of the paper is organized as follows. Preliminary mathematical notation and definitions are given in Section 2. Section 3 discusses the variant of Sparse Vector Technique with Gaussians [56]. Section 4 introduces the syntax and semantics of DiPGauss, the language we use to specify differential privacy algorithms. Section 5 presents VerifyDP$_\varrho$, assuming that there is an algorithm that computes ComputeProb$_\varrho(\cdot)$. It also presents the rephrasing of differential privacy. Section 6 shows how ComputeProb$_\varrho(\cdot)$ can be implemented and presents the integral optimization discussed above. We also present our decidability results here in this section. Section 7 discusses DiPApprox and presents the experimental evaluation. Section 8 discusses related work, and our conclusions are presented in Section 9.

2 Preliminaries

We denote the sets of real, rational, natural, and integer numbers by $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{N}$, and $\mathbb{Z}$, respectively, and the Euler constant by e. A *partial function* f from A to B is denoted as $f : A \rightharpoonup B$. We assume that the reader is familiar with probability. For events E and E, Prob$[E]$ represents the probability of E, and Prob$[E|F]$ the conditional probability of E given F.

Gaussian (Normal) Distribution. The one dimensional Gaussian distribution, denoted by $\mathcal{N}(\mu, \sigma)$, is parameterized by the mean $\mu \in \mathbb{R}$ and the standard deviation $\sigma > 0$. Its probability density function (PDF), $f_{\mu,\sigma}(x)$, is defined as:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

Laplace distribution. The Laplace distribution denoted Lap(μ, b) is parameterized by mean μ and the scaling parameter $b \geq 0$. Its probability density function (PDF), $g_{\mu,b}(x)$, is defined as:

$$g_{\mu,b}(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}} \tag{3}$$

The standard deviation of Lap(μ, b) is $\sqrt{2}b$.

Tail Bounds. In our framework, execution probabilities are computed using nested definite integrals. Apriori, integrals may involve ∞ and $-\infty$ as upper and lower limits, respectively. As we will see (Section 6), we shall use tail bounds on probabilities to approximate the integral as a sum of definite integrals with finite limits.

Given a random variable X with finite mean μ and non-zero standard deviation σ and $\mathsf{th} > 0$, we say that the *two-sided tail probability* is

$$\mathsf{tl}(\mathsf{th}, X, \mu, \sigma) = \mathsf{Prob}[|X - \mu| > \mathsf{th} \cdot \sigma].$$

For Laplace distribution, we have that $\mathsf{tl}(\mathsf{th}, X, \mu, \sigma) = e^{-\mathsf{th}\sqrt{2}}$.

If X is a Gaussian (also known as normal) random variable, we obtain the following bounds on the tail probabilities using Chernoff bounds: $\mathsf{tl}(\mathsf{th}, X, \mu, \sigma) \leq 2e^{-\frac{\mathsf{th}^2}{2}}$.

For a general random variable with finite mean μ and standard deviation σ, Chebyschev's inequality yields that $\mathsf{tl}(\mathsf{th}, X, \mu, \sigma) \leq \frac{1}{\mathsf{th}^2}$. Observe that all the bounds on tail probabilities discussed here monotonically decrease to 0 as th tends to ∞.

Approximate Differential Privacy. Differential privacy [30] is a framework that enables statistical analysis of databases containing sensitive personal information while protecting individual privacy. A randomized algorithm P, called a *differential privacy mechanism*, mediates the interaction between a (potentially dishonest) data analyst and a database D. The analyst submits queries requesting aggregate information such as means, and for each query, P computes its response using both the actual database values and random sampling to produce "noisy" answers. While this approach ensures privacy, it comes at the cost of reduced accuracy. The amount of noise added by P is controlled by a *privacy parameter* $\epsilon > 0$.

The framework provides privacy guarantees for all individuals whose information is stored in database D. This is informally captured as follows. Let $D \setminus \{i\}$ denote the database with individual i's information removed. A secure mechanism M ensures that for any individual i in D and any sequence of possible outputs $\overline{o}$, the probability of P producing $\overline{o}$ remains approximately the same whether querying D or $D \setminus \{i\}$.

Let us define this formally. A differential privacy mechanism is a family of programs P_ϵ whose behavior depends on the privacy parameter ϵ; for notational simplicity, we will often drop the subscript and use P to refer to the programs. Given a set $\mathcal{U}$ of inputs and a set $\mathcal{V}$ of outputs, a randomized function P from $\mathcal{U}$ to $\mathcal{V}$ takes an input in $\mathcal{U}$ and returns a distribution over $\mathcal{V}$. For a measurable set $F \subseteq \mathcal{V}$, the probability that the output of P on u is in F is denoted by Prob$[\mathsf{P}(u) \in F]$. We assume that $\mathcal{U}$ is equipped with a binary asymmetric relation $\Phi \subseteq \mathcal{U} \times \mathcal{U}$, called the *adjacency relation. Adjacent* inputs $(u_1, u_2) \in \Phi$ represent query outputs for databases D and $D \setminus \{i\}$ where i is some individual.

Definition 1. Let $\epsilon, \epsilon_{\mathsf{prv}} > 0$, $0 \leq \delta \leq 1$ and $\Phi \subseteq \mathcal{U} \times \mathcal{U}$ be an adjacency relation. Let P be a randomized algorithm depending on a privacy parameter ϵ with inputs $\mathcal{U}$ and outputs $\mathcal{V}$. We say that P is $(\epsilon_{\mathsf{prv}}, \delta)$-differentially private with respect to Φ if for all measurable subsets $S \subseteq \mathcal{V}$ and $u, u' \in \mathcal{U}$ such that $(u, u') \in \Phi$,

$$\mathsf{Prob}[\mathsf{P}(u) \in S] \leq e^{\epsilon_{\mathsf{prv}}} \mathsf{Prob}[\mathsf{P}(u') \in S] + \delta.$$

ϵ_{prv} is called the *privacy budget*, and δ the *error parameter.*

We say that P is $(\epsilon_{\mathsf{prv}}, \delta)$-strictly differentially private with respect to Φ if for all measurable subsets $S \subseteq \mathcal{V}$ and $u, u' \in \mathcal{U}$ such

that $(u, u') \in \Phi$,

$$\text{Prob}[\text{P}(u) \in S] < e^{\epsilon_{\text{prv}}} \text{Prob}[\text{P}(u') \in S] + \delta.$$

3 Motivating Example: Sparse Vector Technique with Gaussians (SVT-Gauss)

Let us walk through a simple example to demonstrate our method before exploring the full mathematical details. The Sparse Vector Technique (SVT) is a fundamental algorithmic tool in differential privacy that plays an important role in adaptive data analysis and model-agnostic private learning [32, 33]. While the original Sparse Vector Technique (SVT) uses Laplace noise to achieve $(\epsilon_{\text{prv}}, 0)$-differential privacy, recent work has explored a Gaussian variant that achieves $(\epsilon_{\text{prv}}, \delta)$-differential privacy [56]. SVT with Gaussian distribution (SVT-Gauss) offers better utility than the version using Laplace noise through more concentrated noise [56].

Algorithm 1: SVT-Gauss

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow 1$
    exit
  else
    $out[i] \leftarrow 0$
  end
end

The SVT mechanism with Gaussian sampling (SVT-Gauss) is shown in Algorithm 1. Given an array q containing answers to N queries and threshold T, the goal is to output the index of the first query that exceeds the threshold in a privacy preserving manner [4]. The algorithm perturbs the threshold and each query answer by adding Gaussian noise. The algorithm progressively compares the perturbed query answer against the perturbed threshold, assigning 0 to the output array *out* if the query is less and 1 if it is not. The algorithm stops when either 1 is assigned or if all the query answers in q are processed. ϵ is the privacy parameter of the algorithm.

The input set $\mathcal{U}$ consists of N-length vectors q, where the kth element $q[k]$ represents the answer to the kth query on the original database. The adjacency relation Φ on inputs is defined as follows: q_1 and q_2 are adjacent if and only if $|q_1[i] - q_2[i]| \leq 1$ for each $1 \leq i \leq N$.

Recall that the pdf of a Gaussian random variable $\mathcal{N}(\mu, \sigma)$ is denoted as $f_{\mu,\sigma}$. Consider the case when $T = 0$, $N = 2$ and entries in q are limited to $\{0, 1\}$. Thus, there are 4 possible inputs ($[0, 0]$, $[0, 1]$, $[1, 1]$, and $[1, 0]$) and three possible outputs ($[0, 0]$, $[1, 0]$, and $[0, 1]$). Let X_T, X_1, X_2 be the random variables denoting the values of variables $\mathbf{r}_T$, $\mathbf{r}$ during different iterations, in Algorithm 1. Observe that X_T is drawn from $\mathcal{N}(0, \frac{2}{\epsilon})$ and X_i from $\mathcal{N}(q[i], \frac{4}{\epsilon})$.

Consider adjacent inputs $[0, 1]$ and $[1, 1]$. On input $q = [0, 1]$, the probability of output $[0, 0]$ is given by $\text{Prob}[X_1 < X_T, X_2 < X_T]$, which can be computed as:

$$p_1(\epsilon) = \int_{-\infty}^{\infty} f_{0,\frac{2}{\epsilon}}(x_T) \int_{-\infty}^{x_T} f_{0,\frac{4}{\epsilon}}(x_1) \int_{-\infty}^{x_T} f_{1,\frac{4}{\epsilon}}(x_2)\, dx_2\, dx_1\, dx_T$$

Similarly, the probability of output $[0, 1]$ on input $[0, 1]$ is given by $\text{Prob}[X_1 < X_T, X_2 > X_T]$, which can be computed as:

$$p_2(\epsilon) = \int_{-\infty}^{\infty} f_{0,\frac{2}{\epsilon}}(x_T) \int_{-\infty}^{x_T} f_{0,\frac{4}{\epsilon}}(x_1) \int_{x_T}^{\infty} f_{1,\frac{4}{\epsilon}}(x_2)\, dx_2\, dx_1\, dx_T$$

In the same way, when the input is $[1, 1]$, the probability of output $[0, 0]$ and $[0, 1]$ is given by:

$$p_1'(\epsilon) = \int_{-\infty}^{\infty} f_{0,\frac{2}{\epsilon}}(x_T) \int_{-\infty}^{x_T} f_{1,\frac{4}{\epsilon}}(x_1) \int_{-\infty}^{x_T} f_{1,\frac{4}{\epsilon}}(x_2)\, dx_2\, dx_1\, dx_T$$

$$p_2'(\epsilon) = \int_{-\infty}^{\infty} f_{0,\frac{2}{\epsilon}}(x_T) \int_{-\infty}^{x_T} f_{1,\frac{4}{\epsilon}}(x_1) \int_{x_T}^{\infty} f_{1,\frac{4}{\epsilon}}(x_2)\, dx_2\, dx_1\, dx_T$$

Observe that $p_1(\epsilon)$, $p_2(\epsilon)$, $p_1'(\epsilon)$ and $p_2'(\epsilon)$ are functions of ϵ. To check if the adjacent inputs $[0, 1]$ and $[1, 1]$ satisfy the conditions of $(\epsilon_{\text{prv}}, \delta)$-differential privacy for given privacy budget ϵ_{prv}, error δ and output set $\{[0, 0], [0, 1]\}$, we need the following to hold.

$$p_1(\epsilon) + p_2(\epsilon) \leq e^{\epsilon_{\text{prv}}}[p_1'(\epsilon) + p_2'(\epsilon)] + \delta,$$
$$p_1'(\epsilon) + p_2'(\epsilon) \leq e^{\epsilon_{\text{prv}}}[p_1(\epsilon) + p_2(\epsilon)] + \delta.$$

Note that since outputting $[0, 0]$ and $[0, 1]$ are independent events, we can sum their probabilities to obtain an expression for $\{[0, 0], [0, 1]\}$.

Verifying $(\epsilon_{\text{prv}}, \delta)$-differential privacy for a given $\epsilon > 0$ involves computing expressions like $p_i(\epsilon)$ and $p_i'(\epsilon)$ and checking if inequalities like the one above hold for all possible sets of outputs.

The following theorem states the differential privacy of Algorithm 1, and follows from the results of [56].

Theorem 1. *For any $\epsilon > 0$ and $0 < \delta \leq \frac{1}{1+N}$, SVT-Gauss (Algorithm 1) is $(\epsilon_{\text{prv}}, \delta)$-differential privacy for any ϵ_{prv} such that*

$$\epsilon_{\text{prv}} \geq \frac{5\epsilon^2}{32} + \frac{\sqrt{5}}{2}\epsilon\sqrt{\log\frac{1}{\delta}}.$$

SVT-Gauss belongs to the class of programs that we consider in this paper. Observe that when $\epsilon = 0.5, \delta = 0.01, N < 100$, $\epsilon_{\text{prv}} \geq 1.24$. In our experiments, we are able to automatically verify differential privacy with these values of $\epsilon, \epsilon_{\text{prv}}$ and $N \leq 5$. When we consider only single pair of adjacent inputs, our tool is able to handle N upto 25.

4 Program syntax and semantics

We introduce a class of probabilistic programs called DiPGauss, where variables can be assigned values drawn from Gaussian distributions or Laplace distributions, commonly used in differential privacy algorithms. DiPGauss is designed with syntactic restrictions that simplify its encoding into integral expressions. While these restrictions impose certain limitations, they also enable definitive verification of whether a program satisfies differential privacy.

[4] In general, the SVT protocols identifies the first c queries that exceed the threshold, for some fixed parameter c. Also, the privacy of the algorithm can only be guaranteed for query answers that are *sensitive* upto some parameter Δ. Both c and Δ influence the noise that is added when processing the array q. In Algorithm 1, we assume that $c = 1$ and $\Delta = 1$.

Despite its constraints, DiPGauss is a powerful language capable of expressing interesting differentially private algorithms.

Before we present our syntax formally, we observe that differential privacy algorithms are often described as parameterized programs. Colloquially, this parameter is the privacy budget. However, in many instances, the parameter may be different from the privacy budget. (See Theorem 1, Section 3.) Thus, to explicitly distinguish the parameter in the program and the privacy budget, we shall refer to the program parameter as privacy parameter and denote it as ϵ. We will use ϵ_{prv} to denote the privacy budget.

Expressions ($\mathbf{r} \in \mathcal{R}, \mathbf{x} \in \mathcal{X}, q \in \mathbb{Q}, \sim \in \{=, \leq, <, \geq, >, \neq\}$):

$$\begin{array}{rcl} R & := & \mathbf{r} \mid qR \mid R + R \mid R + q \\ B & := & R \sim R \mid R \sim \mathbf{x} \mid \mathbf{x} \sim \mathbf{x} \end{array}$$

Program Statements ($d \in \mathsf{DOM}, a \in \mathbb{Q}^{>0}$):

$$\begin{array}{rcl} S & := & \mathbf{x} \leftarrow d \\ & \mid & \mathbf{r} \leftarrow \mathcal{N}(\mathbf{x}, \frac{a}{\epsilon}) \\ & \mid & \mathbf{r} \leftarrow \mathsf{Lap}(\mathbf{x}, \frac{a}{\epsilon}) \\ & \mid & \mathbf{r} \leftarrow R \\ & \mid & \text{if } B \text{ then } S \text{ else } S \text{ end} \\ & \mid & \mathsf{skip} \\ & \mid & S; S \end{array}$$

Figure 1: **BNF grammar for DiPGauss.** DOM **is a finite discrete domain, taken to be a finite subset of the rationals.** $\mathcal{R}$ **is the set of real random variables and** $\mathcal{X}$ **is the set of** DOM **variables.** $\mathbb{Q}^{>0}$ **denotes set of positive rational numbers.**

4.1 Syntax of DiPGauss Programs

The formal syntax of DiPGauss programs is presented in Figure 1. DiPGauss programs are parametrized, loop-free programs that can sample from continuous distributions. Programs have two types of variables: real random variables and finite-domain variables from DOM, denoted by sets $\mathcal{R}$ and $\mathcal{X}$ respectively. We assume that DOM is some finite subset of rationals and so they can be compared against each other and with real values. Boolean expressions (B) can be constructed by comparing real variables with each other, with DOM-variables, or by comparing DOM variables with each other.

A program is a sequence of statements that can either be assignments to program variables or if conditionals. Assignments can either assign constants (real or DOM values) or values drawn from continuous distributions. In Figure 1, the only distributions we have listed are the Laplace or Gaussian distributions. This is done to keep the presentation simple in this paper. Our results apply even when the syntax of the program language is extended, where samples are drawn from any continuous distribution with a finite mean and variance and a computable probability density function.

Remark. A couple of remarks on the program syntax are in order at this point. DiPGauss does not natively support loops but for-loops can be seen as syntactic sugar in the standard way. A loop of the form *for* $i = 1$ *to* N *do* S can be expanded into a sequence $S_1; S_2; \ldots; S_N$, where N is a constant and each iteration is explicitly unrolled.

Next, Boolean expressions used in conditionals are restricted to comparison between program variables and constants; the syntax does not allow the use of standard logical operators such as negation, conjunction, and disjunction. However, this is not a restriction in expressive power. Taking a step based on the negation of a condition holding can be handled through the else branch, conjunction through nested if-thens, and disjunction through a combination of nesting and else branches.

A program P in DiPGauss is defined as a triple $(\mathcal{I}, \mathcal{O}, S)$, where:

- $\mathcal{I} \subseteq \mathcal{X}$ is a set of private input variables.
- $\mathcal{O} \subseteq \mathcal{X}$ is a set of public output variables.
- $\mathcal{I} \cap \mathcal{O} = \emptyset$
- S is a program statement generated by the non-terminal S of the grammar in Figure 1.

As seen in the grammar of Figure 1, each sampled probability distribution used in the statements of P, has a parameter ϵ. Thus, P is a parameterized program with parameter ϵ appearing in it. The parameter ϵ will be instantiated when computing the probabilities associated with P.

Remark. Strictly speaking, P represents a family of programs, and it is more accurate to represent it as P_ϵ. However, we choose to not mention ϵ explicitly to reduce notational overhead.

Example 1. Algorithm 1 can be rewritten as a DiPGauss program when $T = 0$ and $N = 2$. This is shown as Algorithm 2, where the bounded for-loop is unrolled and written without any loops.

Algorithm 2: SVT-Gauss with $N = 2$ written in DiPGauss

```
Input: q_1, q_2
Output: out_1, out_2
T ← 0
out_1 ← 0
out_2 ← 0
r_T ← N(T, 2/ε)
r_1 ← N(q_1, 4/ε)
if r_1 ≥ r_T then
    out_1 ← 1
else
    r_2 ← N(q_2, 4/ε)
    if r_2 ≥ r_T then
        out_2 ← 1
    end
end
```

We conclude this section with a couple of assumptions about programs in DiPGauss. We will assume that in every program, each real variable is assigned a value at most once along every control path. Clearly, since our program are loop-free, this is not a restriction, as a program where variables are assigned multiple times can be transformed into one that satisfies this assumption by introducing new variables. However, making this assumption about our programs will make it easier to describe the semantics.

Finally, we will assume that programs are *well-formed*. That is, all references in non-input variables in the program are preceded by assignments to those variables.

4.2 Semantics

The semantics for DiPGauss programs presented in this section crucially relies on the notion of state. Typically, state for a non-recursive imperative program without dynamic allocation is just an assignment of values to the program variables. However, for programs where real variables are assigned values from continuous distributions, this does not work. Instead, we will record the values of real variables "symbolically" — we will either record the distribution (plus parameters like mean) from which the value of a real variable is sampled or the expression it is assigned [5]. However, to reliably assign probabilities to paths with such "symbolic states", we also need to track the Boolean conditions that are assumed to hold so far. Based on these intuitions let us define states formally.

States. Let BExp and RExp denote the set of expressions derived from the non-terminals B and R, respectively, in Figure 1. Define $\mathsf{Distr} = \{\mathsf{Gaussian}, \mathsf{Laplace}\} \times \mathbb{Q} \times \mathbb{Q}^{>0}$; elements of this set denote distributions along with appropriate parameters. So, $(\mathsf{Gaussian}, \mu, \sigma)$ represents the Gaussian distribution with mean μ and standard deviation σ while $(\mathsf{Laplace}, \mu, b)$ represents the Laplace distribution with mean μ and scaling parameter b. A *state* st is then a triple $\mathsf{st} = (\alpha, \beta, G)$, where $\alpha : \mathcal{X} \rightharpoonup \mathsf{DOM}$, $\beta : \mathcal{R} \rightharpoonup \mathsf{Distr} \cup \mathsf{RExp}$, and $G \subseteq \mathsf{BExp}$. Here α is a partial map that assigns a value to DOM variables, β is a partial function mapping real variables to either the distribution from which they are sampled or the expression that is assigned to them, and G is a set of Boolean conditions.

For a state st and a variable $\mathbf{r} \in \mathcal{R}$, we say that $\mathbf{r}$ is an *independent* variable if $\beta(\mathbf{r}) \in \mathsf{Distr}$ and a *dependent* variable if $\beta(\mathbf{r}) \in \mathsf{RExp}$. For a state st, (DOM or real) variable $\mathbf{v}$ and value $u \in \mathsf{DOM} \cup \mathsf{Distr} \cup \mathsf{RExp}$, $\mathsf{st}[\mathbf{v} \mapsto u]$ denotes the state that maps $\mathbf{v}$ to u and is otherwise identical to st.

Final States. $[\![\mathsf{P}]\!]_{\mathsf{st}}$ denotes the set of *final states* reached when P is executed from starting state st. It is inductively defined as follows.

- $[\![\mathsf{skip}]\!]_{\mathsf{st}} = \{\mathsf{st}\}$.
- $[\![\mathbf{x} \leftarrow d]\!]_{\mathsf{st}} = \{\mathsf{st}[\mathbf{x} \mapsto d]\}$.
- $[\![\mathbf{r} \leftarrow \mathcal{N}(\mathbf{x}, \frac{a}{\epsilon})]\!]_{\mathsf{st}} = \left\{\mathsf{st}\left[\mathbf{r} \mapsto \left(\mathsf{Gaussian}, \mathsf{st}(\mathbf{x}), \frac{a}{\epsilon}\right)\right]\right\}$.
- $[\![\mathbf{r} \leftarrow \mathsf{Lap}(\mathbf{x}, \frac{a}{\epsilon})]\!]_{\mathsf{st}} = \left\{\mathsf{st}\left[\mathbf{r} \mapsto \left(\mathsf{Laplace}, \mathsf{st}(\mathbf{x}), \frac{a}{\epsilon}\right)\right]\right\}$.
- $[\![\mathbf{r} \leftarrow R]\!]_{\mathsf{st}} = \{\mathsf{st}\,[\mathbf{r} \mapsto R']\}$, where R' is the expression obtained by replacing every dependent $\mathbf{r}' \in \mathcal{R}$ that appears in R with $\beta(\mathbf{r}')$.
- $[\![\mathsf{if}\ B\ \mathsf{then}\ \mathsf{P}_1\ \mathsf{else}\ \mathsf{P}_2\ \mathsf{end}]\!]_{\mathsf{st}} = [\![\mathsf{P}_1]\!]_{\mathsf{st}_{\{B\}}} \cup [\![\mathsf{P}_2]\!]_{\mathsf{st}_{\{\neg B\}}}$, where $\mathsf{st}_{\{B\}} = \mathsf{st}[G \mapsto \mathsf{st}(G) \cup \{B\}]$ and $\mathsf{st}_{\{B\}} = \mathsf{st}[G \mapsto \mathsf{st}(G) \cup \{\neg B\}]$. Here, $\neg B$ denotes the "flipped" comparison: for $\sim \in \{<, \leq, >, \geq, =, \neq\}$, $\neg(R \sim R') \triangleq (R \overline{\sim} R')$, where $\overline{<} = \geq$, $\overline{\leq} = >$, $\overline{>} = \leq$, $\overline{\geq} = <$, $\overline{=} = \neq$, and $\overline{\neq} = ==$.
- $[\![\mathsf{P}_1; \mathsf{P}_2]\!]_{\mathsf{st}} = \bigcup_{\tau \in [\![\mathsf{P}_1]\!]_{\mathsf{st}}} \{\tau' \mid \tau' \in [\![\mathsf{P}_2]\!]_{\tau}\}$.

Input/Output Behavior. Let us fix a program $\mathsf{P} = (\mathcal{I}, \mathcal{O}, S)$, an input valuation $u : \mathcal{I} \to \mathsf{DOM}$ and output valuation $o : \mathcal{O} \to \mathsf{DOM}$. The *final states of* P *on input* u *with output* o, denoted $\rho(u, o, \mathsf{P})$, is defined as

$$\rho(u, o, \mathsf{P}) = \{(\alpha, \beta, G) \in [\![\mathsf{P}]\!]_{\mathsf{st}_0} \mid \forall y \in \mathcal{O}, \alpha(y) = o(y)\}$$

[5] Recall that we assume that every real variable is assigned at most once during an execution; see Section 4.1.

where the initial state $\mathsf{st}_0 = (\alpha_0, \beta_0, G_0)$ has $G_0 = \emptyset$, β_0 is the partial function with empty domain, and α_0 is the partial function with domain $\mathcal{I}$ such that $\alpha_0(x) = u(x)$ for $x \in \mathcal{I}$.

Probability of a state. Let $\tau = (\alpha, \beta, G)$ be a state. Define

$$G_{\mathsf{const}} = \{g \in G \mid g = \mathbf{x} \sim \mathbf{x}',\ \text{where } \mathbf{x}, \mathbf{x}' \in \mathcal{X}\},$$

and

$$G_{\mathsf{rand}} = \{g \in G \mid g = \mathbf{r} \sim \mathbf{x} \text{ or } g = \mathbf{r} \sim \mathbf{r}' \text{ where } \mathbf{r}, \mathbf{r}' \in \mathcal{R}, \mathbf{x} \in \mathcal{X}\}.$$

Let $\mathsf{eval}_{\mathsf{c}}(G_{\mathsf{const}})$ be the Boolean value given by

$$\mathsf{eval}_{\mathsf{c}}(G_{\mathsf{const}}) = \bigwedge_{(\mathbf{x} \sim \mathbf{x}') \in G_{\mathsf{const}}} \alpha(\mathbf{x}) \sim \alpha(\mathbf{x}').$$

In the above equation, when $G_{\mathsf{const}} = \emptyset$, the conjunction is taken to be true as is standard.

Let rand be the partial function on $\mathcal{R}$ with the same domain as β defined as follows. If $\beta(\mathbf{r}) = (\mathsf{Gaussian}, \mu, \sigma)$, then $\mathsf{rand}(\mathbf{r})$ is the Gaussian random variable $X_{\mathbf{r}}$ with parameters (μ, σ). If $\beta(\mathbf{r}) = (\mathsf{Laplace}, \mu, b)$, then $\mathsf{rand}(\mathbf{r})$ is the Laplace random variable $X_{\mathbf{r}}$ with parameters (μ, b). If $\beta(\mathbf{r}) = R$, where $R \in \mathsf{RExp}$, then $\mathsf{rand}(\mathbf{r})$ is the expression obtained from R by replacing every independent variable $\mathbf{r}'$ appearing in R by the random variable $X_{\mathbf{r}'}$. Now, let us define

$$\mathsf{eval}_{\mathsf{r}}(G_{\mathsf{rand}}) = \left(\bigwedge_{\mathbf{r} \sim \mathbf{r}' \in G_{\mathsf{rand}}:\ \mathbf{r}, \mathbf{r}' \in \mathcal{R}} \mathsf{rand}(\mathbf{r}) \sim \mathsf{rand}(\mathbf{r}')\right) \wedge \left(\bigwedge_{\mathbf{r} \sim \mathbf{x} \in G_{\mathsf{rand}}:\ \mathbf{r} \in \mathcal{R},\ \mathbf{x} \in \mathcal{X}} \mathsf{rand}(\mathbf{r}) \sim \alpha(\mathbf{x})\right)$$

Now, for a given value to the parameter ϵ, the probability of τ is given by:

$$\mathsf{Prob}[\epsilon, \tau] = \begin{cases} 0 & \text{if } \mathsf{eval}_{\mathsf{c}}(G_{\mathsf{const}}) = \mathsf{false} \\ \mathsf{Prob}[\epsilon, \mathsf{eval}_{\mathsf{r}}(G_{\mathsf{rand}})] & \text{otherwise} \end{cases}$$

where $\mathsf{Prob}[(\epsilon, \mathsf{eval}_{\mathsf{r}}(G_{\mathsf{rand}}))]$ is the probability that the random variables $X_{\mathbf{r}}$ satisfy the condition $\mathsf{eval}_{\mathsf{r}}(G_{\mathsf{rand}})$ for the given value of ϵ.

Probability of Output. For any given $\epsilon > 0$, the probability that the program P outputs the valuation o, with input values given by valuation u, denoted by $\mathsf{Prob}[\epsilon, u, o, \mathsf{P}]$, is defined as

$$\mathsf{Prob}[\epsilon, u, o, \mathsf{P}] = \sum_{\tau \in \rho(u, o, \mathsf{P})} \mathsf{Prob}[\epsilon, \tau]$$

Example 2. For the SVT-Gauss program given in Algorithm 2, called P here, the set of input variables $\mathcal{I} = \{q_1, q_2\}$, and the set of output variables $\mathcal{O} = \{out_1, out_2\}$.

Consider an input assignment $u = \{q_1 \mapsto 0, q_2 \mapsto 1\}$ and an output assignment $o = \{out_1 \mapsto 0, out_2 \mapsto 1\}$. In this case, it can be easily seen that there is a single state $\tau = (\alpha, \beta, G)$ in $\rho(u, o, \mathsf{P})$ where $\alpha(out_1) = 0$, $\alpha(out_2) = 1$ and $G = G_{\mathsf{rand}} = \{\mathbf{r}_1 < \mathbf{r}_T, \mathbf{r}_2 \geq \mathbf{r}_T\}$.

Now, we have

$$\mathsf{Prob}[\epsilon, u, o, \mathsf{P}] = \mathsf{Prob}[\epsilon, \tau] = \mathsf{Prob}[X_{\mathbf{r}_1} < X_{\mathbf{r}_T} \wedge X_{\mathbf{r}_2} \geq X_{\mathbf{r}_T}]$$

For a set of output valuations F, the probability of P producing an output in F on input u can be defined as $\sum_{o \in F} \mathsf{Prob}[\epsilon, u, o, \mathsf{P}]$. Using this, the definitions $(\epsilon_{\mathsf{prv}}, \delta)$-differential privacy and $(\epsilon_{\mathsf{prv}}, \delta)$-strict differential privacy given in Definition 1, can be precisely instantiated for DiPGauss programs.

Definition 2. The *differential privacy problem* is the following: Given a DiPGauss program P, rational numbers $\epsilon_0 > 0$, $\epsilon_{\text{prv}} > 0$, $\delta \in [0, 1]$, determine if P with privacy parameter taking value ϵ_0 is $(\epsilon_{\text{prv}}, \delta)$-differentially private.

5 Checking differential privacy for DiPGauss programs

We describe our core algorithms for checking differential privacy of programs. For the rest of the section, we assume that we are given a DiPGauss program $\mathsf{P} = (\mathcal{I}, \mathcal{O}, S)$ with privacy parameter ϵ. Let $\mathcal{U}$ be the set of possible functions from $\mathcal{I}$ to DOM, and $\mathcal{V}$ be the set of possible functions from $\mathcal{O}$ to DOM respectively, ϵ_{prv} denote the privacy budget. Let δ denote the error parameter.

Definition 3. A precision ϱ is a natural number. A ϱ-approximation of a real number p is an interval $[L, U]$ such that L, U are rational numbers, $L \leq p \leq U$ and $U - L \leq 2^{-\varrho}$.

Verifying differential privacy of P requires checking inequalities across all subsets of possible outputs, as specified in Definition 1. Two key challenges arise in this context. We describe the challenges below and develop two key innovations to tackle them.

(1) Apriori, the inequality in Definition 1 needs to be checked exponentially many times as there are $2^{|\mathcal{O}|}$ possible subsets of outputs. For example, consider Algorithm 1 with N inputs: we have N possible outputs, resulting in 2^N subsets to check for each adjacent pair. Given that we have to check *all* adjacent pairs, this makes these checks even more expensive. Instead of checking every possible subset, we rephrase the differential privacy definition so that only one equation needs to be checked for each adjacent input (See Lemma 2).
(2) As mentioned in the Introduction, it is unclear that $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ can be computed exactly. Hence, instead of computing $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ exactly, we compute ϱ-approximations of $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ and $e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u, o, \mathsf{P}]$ for a given precision ϱ. This allows us to design an algorithm, VerifyDP_ϱ that returns three possible values: DP, Not_DP, and Unknown. The algorithm is sound in that if it returns DP (Not_DP), then the input program P is differentially private (not differentially private, respectively).

The following lemma whose proof is given in Appendix A allows us to tame the exponential number of subsets of outputs in the differential privacy checks.

Lemma 2. *A DiPGauss program* P *is* $(\epsilon_{\text{prv}}, \delta)$*-differentially private (for* $\epsilon_{\text{prv}} > 0$ *and* $\delta \in [0, 1]$*) with respect to* Φ *iff for all* $(u, u') \in \Phi$,

$$\delta_{u,u'} = \sum_{o \in \mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \mathsf{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \mathsf{P}], 0) \leq \delta \quad (4)$$

It will be useful to consider the set of error parameters for which Equation 4 becomes an equality.

Definition 4. For a program P_ϵ with adjacency Φ, the set of *critical* error parameters is defined to be the set

$$\mathsf{D}_{\mathsf{P},\epsilon,\epsilon_{\text{prv}},\Phi} = \{\delta_{u,u'} \mid (u, u') \in \Phi \,\&\, \delta_{u,u'} = \sum_{o \in \mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \mathsf{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \mathsf{P}], 0)\}.$$

We shall now describe the VerifyDP_ϱ algorithm that allows us to verify (soundly) differential and non-differential privacy.

5.1 VerifyDP algorithm

We will assume that we can approximate $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ and $e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u, o, \mathsf{P}]$ to any desired degree of precision. We shall refer to $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ as output probability and $e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u, o, \mathsf{P}]$ as scaled output probability.

Definition 5. We say that the output probability is effectively approximable if there is an algorithm $\text{ComputeProb}_\varrho(\cdot)$ such that on input ϱ, rational number $\epsilon > 0$, $u \in \mathcal{U}, o \in \mathcal{V}$, and DiPGauss program P, $\text{ComputeProb}_\varrho(\epsilon, u, o, \mathsf{P})$ outputs a ϱ-approximation of $\text{Prob}[\epsilon, u, o, \mathsf{P}]$.

We say that the scaled output probability is effectively approximable if there $\text{ComputeScaleProb}_\varrho(\cdot)$ such that on input ϱ, rational $\epsilon_{\text{prv}}, \epsilon > 0$, $u \in \mathcal{U}, o \in \mathcal{V}$ and DiPGauss program P, $\text{ComputeScaleProb}_\varrho(\epsilon_{\text{prv}}, \epsilon, u, o, \mathsf{P}, \epsilon_{\text{prv}})$ outputs a ϱ-approximation of $e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u, o, \mathsf{P}]$, respectively.

If the algorithm $\text{ComputeProb}_\varrho(\cdot)$ ($\text{ComputeScaleProb}_\varrho(\cdot)$, respectively) computes ϱ-approximation for $\text{Prob}[\epsilon, u, o, \mathsf{P}]$ ($e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u, o, \mathsf{P}]$, respectively) for a specific ϱ only (and not for all ϱ), we say that the the output probability (scaled output probability, respectively) is effectively ϱ-approximable.

We have the following.

Proposition 3. *If the output probability is effectively approximable then the scaled output probability is effectively approximable.*

The VerifyDP_ϱ algorithm (See Algorithm 3) checks differential privacy for $\mathsf{P} = (\mathcal{I}, \mathcal{O}, S)$ for all input pairs given by the adjacency relation Φ, privacy parameter ϵ, error parameter δ and privacy budget ϵ_{prv}. The algorithm assumes the existence of $\text{ComputeProb}_\varrho(\cdot)$, $\text{ComputeScaleProb}_\varrho(\cdot)$, and proceeds as follows.

The algorithm processes one adjacent pair at a time. The algorithm also maintains a flag b. Intuitively, the flag b is true if P is differentially private for all input pairs u and u' checked thus far. For each adjacent pair $(u, u') \in \Phi$, and each output $o \in \mathcal{V}$, the algorithm calls $\text{ComputeProb}_\varrho(\epsilon, u, o, \mathsf{P})$, and $\text{ComputeScaleProb}_\varrho(\epsilon_{\text{prv}}, \epsilon, u', o, \mathsf{P})$. The resulting values are stored in dictionaries store and store_scale. The dictionary store stores the output probabilities, and the dictionary store_scale stores the scaled output probabilities. Once the probabilities for each output o have been stored, the algorithm calls the function VerifyPair to either prove or disprove differential privacy for the input pair u and u', or to indicate that the current precision ϱ is insufficient (i.e., the result is Unknown).

If VerifyPair returns Not_DP for any input pair, the flag b is set to false and the algorithm terminates immediately, concluding that the program is not differentially private. In cases where VerifyPair returns Unknown for a particular pair, the flag b is set to false. However, we continue to process the remaining pairs in Φ. The rationale is that we may be able to conclude that P is not differentially private for some other pair that has not been checked as yet.

After processing all adjacent pairs, if the flag b remains true, the algorithm returns that the program is differentially private; otherwise, it returns Unknown.

Algorithm 3: VerifyDP$_\varrho$

Input: Program P, Adjacency Φ, Privacy parameter ϵ, Error parameter δ, Privacy Budget ϵ_{prv}, precision ϱ

Output: One of (a) DP (satisfies DP) (b) Not_DP (violates DP) (c) Unknown

```
store ← ∅
b ← true
foreach (u, u′) ∈ Φ do
    foreach o ∈ 𝒱 do
        if (o, u) ∉ store then
            store[(o, u)] ← ComputeProb_ϱ(ϵ, u, o, P)
        end
        if (o, u′) ∉ store_scale then
            store_scale[(o, u′)] ←
              ComputeScaleProb_ϱ(ϵ_prv, ϵ, u, o, P)
        end
    end
    res = VerifyPair(u, u′, δ, store, store_scale)
    if res = Not_DP then
        b ← false
        return Not_DP
    end
    if res = Unknown then
        b ← false
    end
end
if b then
    return DP
end
return Unknown
```

The VerifyPair function

We now discuss the VerifyPair function (Algorithm 4). Given an adjacent input pair u, u', the error parameter δ, the dictionaries store and store_scale, and precision ϱ, VerifyPair is tasked with proving or disproving differential privacy.

The function iterates over the set of outputs. For each output o, it computes an upper bound and lower bound on $\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)$ as follows. For each output o, it retrieves that intervals $I_u = [L_1, U_1]$ and $I_{u'} = [L_2, U_2]$, the intervals containing the $\text{Prob}[\epsilon, u, o, \text{P}]$ and $e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}]$ respectively. Now, $\max(U_1 - L_2, 0)$ ($\max(L_1 - U_2, 0)$, respectively) can be seen to be an upper bound (lower bound, respectively) on $\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)$.

The upper bounds and lower bounds on $\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)$ are accumulated in Δ_{max} and Δ_{min} respectively. Once the iteration over the set of outputs is over, VerifyPair declares that P is differentially private for u, u' if $\Delta_{max} \leq \delta$ and not differentially private if $\Delta_{min} > \delta$. If neither $\Delta_{max} \leq \delta$ not $\Delta_{min} > \delta$, then VerifyPair returns Unknown.

Algorithm 4: VerifyPair

```
Function VerifyPair(u, u′, δ, store, store_scale)
1   Δ_min ← 0
2   Δ_max ← 0
3   foreach o ∈ 𝒱 do
4       I_u ← store[(o, u)]
5       I_u′ ← store_scale[(o, u′)]
6       L_1, U_1 ← LB(I_u), UB(I_u)
7       L_2, U_2 ← LB(I_u′), UB(I_u′)
8       Δ_max ← Δ_max + max(U_1 − L_2, 0)
9       Δ_min ← Δ_min + max(L_1 − U_2, 0)
    end
10  if Δ_max ≤ δ then
11      return DP
    end
12  if Δ_min > δ then
13      return Not_DP
    end
14  return Unknown
```

5.2 On the soundness and completeness of VerifyDP$_\varrho$

We show that VerifyDP$_\varrho$ always gives a sound answer. Please see Appendix B for the proof.

Lemma 4 (Soundness of VerifyDP$_\varrho$)**.** *Given precision ϱ, let the output and scaled output probabilities be effectively ϱ-approximable. Let $\text{P} = (\mathcal{I}, \mathcal{O}, S)$ be a program with privacy parameter ϵ. Let Φ be an adjacency relation, $\epsilon_{\text{prv}} > 0$ be a privacy budget and $\delta \in [0, 1]$ be an error parameter.*

(1) If VerifyDP$_\varrho$(P, $\Phi, \epsilon, \epsilon_{\text{prv}}, \delta, \epsilon$) *returns Not_DP for precision ϱ then* P *does not satisfy $(\epsilon_{\text{prv}}, \delta)$-differential privacy with respect to Φ.*

(2) If VerifyDP$_\varrho$(P, $\Phi, \epsilon, \epsilon_{\text{prv}}, \delta, \epsilon$) *returns DP for precision ϱ then* P *satisfies $(\epsilon_{\text{prv}}, \delta)$-differential privacy with respect to Φ.*

Now we show that, if P is not differentially private, then VerifyDP$_\varrho$ will return Not_DP for large enough precision, and if P is differentially private then VerifyDP$_\varrho$ will return DP for all non-critical error parameters, for large enough precision. Please see Appendix B for the proof.

Lemma 5 (Completeness of VerifyDP$_\varrho$)**.** *Let the output probability be effectively approximable for all precision ϱ. Let $\text{P} = (\mathcal{I}, \mathcal{O}, S)$ be a program with privacy parameter ϵ. Let Φ be an adjacency relation, $\epsilon_{\text{prv}} > 0$ be a privacy budget and $\delta \in [0, 1]$ be an error parameter.*

(1) If P *does not satisfy $(\epsilon_{\text{prv}}, \delta)$-differential privacy with respect to Φ, then there is a precision ϱ_0 such that* VerifyDP$_\varrho$(P, $\Phi, \epsilon, \epsilon_{\text{prv}}, \delta, \epsilon$) *returns Not_DP for each $\varrho > \varrho_0$.*

(2) If P *satisfies $(\epsilon_{\text{prv}}, \delta)$-differential privacy with respect to Φ, and $\delta \notin \text{D}_{\text{P},\epsilon,\epsilon_{\text{prv}},\Phi}$ (see Definition 4) then there is a precision ϱ_0 such that* VerifyDP$_\varrho$(P, $\Phi, \epsilon, \epsilon_{\text{prv}}, \delta, \epsilon$) *returns DP for each $\varrho > \varrho_0$.*

Suppose, we run VerifyDP$_\varrho$ repeatedly for P by incrementing the precision ϱ until the algorithm returns DP or Not_DP. If P is not differentially private then we are guaranteed to see Not_DP by Lemma 5. If P is differentially private then we will eventually see DP for all non-critical error parameters δ. Thus, if we can show that if output probabilities are effectively approximable, we can conclude that checking non-differential privacy of DiPGauss programs is recursively enumerable, and decidable for all but a finite set of error parameters. This is subject of the next section.

6 Approximating Output Probabilities

Assume that we are given a DiPGauss program $\mathsf{P}_\epsilon = (\mathcal{I}, \mathcal{O}, S)$ with privacy parameter ϵ. We describe the algorithm ComputeProb$_\varrho$ that computes ϱ-approximants of Prob$[\epsilon, u, o, P]$. Fix $\epsilon > 0, u$ and o. Recall that Prob$[\epsilon, u, o, \mathsf{P}]$ (See Section 4), is given by:

$$\mathsf{Prob}[\epsilon, u, o, \mathsf{P}] = \sum_{\tau \in \rho(u,o,\mathsf{P})} \mathsf{Prob}[\epsilon, \tau].$$

For computing the above value, we need to compute Prob$[\epsilon, \tau]$, for each final state $\tau \in \rho(u, o, \mathsf{P})$. Furthermore, observe that since the number of final states of a program is independent of the precision ϱ, it suffices to show that there is an algorithm that produces a ϱ_0-approximant of Prob$[\epsilon, \tau]$ for given ϱ_0.

Fix $\tau = (\alpha, \beta, G)$. As given in Section 4, let G_{const} and G_{rand} correspond to the set of guards in G with comparison of domain variables and comparison of random variables, respectively. Suppose $\mathsf{eval_c}(G_{\mathsf{const}}) = \mathsf{true}$. Then, Prob$[\epsilon, \tau]$ is given by Prob$[\epsilon, \mathsf{eval_r}(G_{\mathsf{rand}})]$. If $\mathsf{eval_c}(G_{\mathsf{const}}) = \mathsf{false}$, Prob$[\epsilon, \tau] = 0$. Since, $\mathsf{eval_c}(G_{\mathsf{const}})$ can be easily computed, it suffices to show that Prob$[\epsilon, \mathsf{eval_r}(G_{\mathsf{rand}})]$ can be computed up-to any precision.

Recall that the variables appearing in the guards of G_{rand} are all independent variables in $\mathcal{R}$. Let $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{n-1}$ denote the variables that appear in the guards of G_{rand}. Also, recall that $X_{\mathbf{r}_0}, X_{\mathbf{r}_1}, \ldots, X_{\mathbf{r}_{n-1}}$ are independent random variables of Laplacian or Gaussian distributions. Let $\mu_0, \mu_1, \ldots, \mu_{n-1}$ denote the means of these random variables, and let $\sigma_0, \sigma_1, \ldots, \sigma_{n-1}$ denote their standard deviations, respectively. Let $h_0, h_1, \ldots, h_{n-1}$ be the corresponding probability density functions of these random variables. Observe that $\mathsf{eval_r}(G_{\mathsf{rand}})$ is a conjunction of linear constraints over $X_{\mathbf{r}_0}, X_{\mathbf{r}_1}, \ldots, X_{\mathbf{r}_{n-1}}$.

We shall exploit the observation that the probability Prob$[\epsilon, \tau]$ can be expressed as sum of nested definite integrals. The primary challenge in exploiting this observation is that such integrals may have ∞ or $-\infty$ as bounds. We will handle this challenge as follows. Given a threshold $\mathsf{th} > 0$, we can write Prob$[\epsilon, \tau]$ as $\mathsf{bpr}(\epsilon, \tau, \mathsf{th}) + \mathsf{tpr}(\epsilon, \tau, \mathsf{th})$. Here $\mathsf{bpr}(\epsilon, \tau, \mathsf{th}) = \mathsf{Prob}[\epsilon, \mathsf{eval_r}(G_{\mathsf{rand}}) \wedge \bigwedge_{0 \le i \le n-1} (\mu_i - \mathsf{th} \cdot \sigma_i \le X_{\mathbf{r}_i} \wedge \{X_{\mathbf{r}_i} \le \mu_i + \mathsf{th} \cdot \sigma_i)\}]$ is the probability of obtaining output o on input u under the constraint that each sampled value $X_{\mathbf{r}_i}$ in τ remains within $\mathsf{th} \cdot \sigma_i$ from μ_i. The term $\mathsf{tpr}(\cdot)$ accounts for the tail probability, capturing cases where at least one sampled value $X_{\mathbf{r}_i}$ deviates by at least $\mathsf{th} \cdot \sigma_i$ from μ_i. As we shall argue shortly, $\mathsf{bpr}(\cdot)$ can be computed to any desired precision. Moreover, by known tail bounds, we can always select th such that $\mathsf{tpr}(\epsilon, \tau, \mathsf{th})$ arbitrarily small. This will guarantee that Prob$[\epsilon, \tau]$ can be computed to any required degree of precision. We have the following observation; proof is in Appendix C.

Lemma 6 (Choosing th)**.** *There is an algorithm that given program* P*, final state τ of* P*, rational number $\epsilon > 0$, and precision ϱ, outputs* th *such that* $0 \le \mathsf{tpr}(\epsilon, \tau, \mathsf{th}) \le \frac{1}{2 \cdot 2^\varrho}$.

6.1 Computing probabilities via integral expressions

The computation of $\mathsf{bpr}(\epsilon, \tau, \mathsf{th})$ for a given threshold th is accomplished by constructing a proper nested definite integral expression.[6] To derive the integral expression, we analyze the set G_{rand} of guards along with the set of the equations $\{\mu_i - \mathsf{th}_i \cdot \sigma_i \le X_{\mathbf{r}_i} \mid 0 \le i \le n-1\} \cup \{X_{\mathbf{r}_i} \le \mu_i + \mathsf{th}_i \cdot \sigma_i \mid 0 \le i \le n-1\}$.

An integral expression $\mathcal{E}$ over $X_{\mathbf{r}_1}, \ldots, X_{\mathbf{r}_n}$ is said to be in *normalized* form if there exists a permutation $\pi(0), \pi(1), ..., \pi(n-1)$ of the indices $0, ..., n-1$ and rational constants θ_0^-, θ_0^+, and *linear* functions $\theta_i^-(y_0, y_1, ..., y_{i-1})$, $\theta_i^+(y_0, y_1, ..., y_{i-1})$ in the variables $y_0, ..., y_{i-1}$, for $1 \le i \le n-1$, such that

$$\mathcal{E} = \int_{\theta_0^-}^{\theta_0^+} h_{\pi(0)}(y_0) \int_{\theta_1^-}^{\theta_1^+} h_{\pi(1)}(y_1) \ldots \int_{\theta_{n-1}^-}^{\theta_{n-1}^+} h_{\pi(n-1)}(y_{n-1}) dy_{n-1} \ldots dy_0 \tag{5}$$

Lemma 7. $\mathsf{bpr}(\epsilon, \tau, \mathsf{th}) = \mathsf{Prob}[\epsilon, \mathsf{eval_r}(G_{\mathsf{rand}}) \wedge \bigwedge_{0 \le i \le n-1} (\mu_i - \mathsf{th} \cdot \sigma_i \le X_{\mathbf{r}_i} \wedge \{X_{\mathbf{r}_i} \le \mu_i + \mathsf{th} \cdot \sigma_i)\}]$ *is a finite sum of integral expressions in* normalized *form over the random variables* $X_{\mathbf{r}_0}, ..., X_{\mathbf{r}_{n-1}}$.

The proof of the above lemma follows from the results of [15] and uses the same approach as that used in the proof of Lemma 18 in [4]. We show that the output probabilities are effectively approximable in the following theorem whose proof is located in Appendix D.

Theorem 8. *The output probabilities are ϱ-approximable. That is, there is an algorithm* ComputeProb$_\varrho(\epsilon, u, o, \mathsf{P})$ *that takes inputs $\varrho, \epsilon, u, o,$* P *and returns a ϱ-approximation of* Prob$[\epsilon, u, o, \mathsf{P}]$.

We get immediately from Lemma 4, Lemma 5 and Theorem 8 that we can automatically check $(\epsilon_{\mathsf{prv}}, \delta)$-differential privacy problem of a DiPGauss program P_ϵ for all non-critical error parameters (See Definition 4).

Theorem 9. *The problem of determining if a DiPGauss program* P_ϵ *is not-$(\epsilon_{\mathsf{prv}}, \delta)$-differentially private with respect to adjacency relation Φ for a given $\epsilon > 0, \epsilon_{\mathsf{prv}} > 0, \delta \in [0, 1]$ is recursively enumerable.*

The problem of checking if P_ϵ *is $(\epsilon_{\mathsf{prv}}, \delta)$-differentially private with respect to adjacency relation Φ is* decidable *for all $\delta \in [0, 1]$,* except *those in the finite set* $\mathsf{D}_{\mathsf{P},\epsilon,\epsilon_{\mathsf{prv}},\Phi}$.

6.2 Optimization of Integral Expressions

We will now present a method for transforming an integral expression $\mathcal{E}$ in normalized form into another equivalent integral expression $\mathcal{E}'$, so that the depth of nesting of $\mathcal{E}'$ is minimal to enable efficient evaluation. Without loss of generality, consider the integral expression $\mathcal{E}$ as given by the Equation 5. To do the transformation of $\mathcal{E}$, we define a directed graph $\mathcal{G}_\mathcal{E}$, called the dependency graph of $\mathcal{E}$, given by $\mathcal{G}_\mathcal{E} = (V, E)$, where $V = \{\mathbf{r}_0, \ldots, \mathbf{r}_{n-1}\}$ and E is the set of edges $(\mathbf{r}_{\pi(\ell)}, \mathbf{r}_{\pi(j)})$ such that y_ℓ appears in either of the linear expressions θ_j^-, θ_j^+, for $0 \le j, \ell < n$.

[6] By a definite proper integral, we mean an integral where the function being integrated is continuous on a bounded finite interval, and both the limits of integration are finite.

Algorithm 5: Optimized Integral Expressions Generation

Input: $\mathcal{G}$ a Directed Acyclic Graph (DAG)
Output: Integral Expressions
Function GenExpr($\mathcal{G}$):
  if $\mathcal{G}$ has WCCs $\mathcal{G}_1, \cdots, \mathcal{G}_n$ **then**
    return $(\text{GenExpr}(\mathcal{G}_1))(\text{GenExpr}(\mathcal{G}_2))\cdots(\text{GenExpr}(\mathcal{G}_n))$
  end
  if $\mathcal{G}$ is singleton node $\mathbf{r}_{\pi(j)}$ for some $j, 0 \le j < k$ **then**
    return $\int_{\theta_j^-}^{\theta_j^+} h_{\pi(j)}(y_j)\, dy_j$
  end
  Let Src $= \{\mathbf{r}_{\pi(j_0)}, \cdots, \mathbf{r}_{\pi(j_{\ell-1})}\}, 0 < \ell \le n$ be source nodes
  return $\int_{\theta_{j_0}^-}^{\theta_{j_0}^+} \cdots \int_{\theta_{j_{\ell-1}}^-}^{\theta_{j_{\ell-1}}^+} \prod_{i=0}^{\ell-1} h_{\pi(j_i)}\, \text{GenExpr}(\mathcal{G}')\, dy_{j_{\ell-1}} \cdots dy_{j_0}$
  where $\mathcal{G}' = \mathcal{G} \setminus$ Src

We propose a method that takes the dependency graph $\mathcal{G}_\mathcal{E}$ of an integral expression as given in the Equation 5 and generates an equivalent nested integral that has a minimal depth of nesting. This method is given by the Algorithm 5. This algorithm consists of a recursive function GenExpr that takes as input a sub-graph of the dependency graph $\mathcal{G}_\mathcal{E}$ of an integral expression as given in Equation 5. Initially the algorithm is invoked with $\mathcal{G}_\mathcal{E}$ as the argument. The permutation π used in the algorithm, in all recursive invocations, is the permutation π that is used in Equation 5.

The algorithm works as follows. Initially, it checks whether the graph $\mathcal{G}$ has multiple Weakly Connected Components, in short WCCs [7]. If $\mathcal{G}$ has more than one WCC, the algorithm is invoked recursively for each WCC, and the product of the resulting expressions is returned. Observe that the depth of nesting of the resulting integral expression is the maximum of the depths of integral expressions for the individual WCCs. Next, if the graph $\mathcal{G}$ consists of a single node $\mathbf{r}_{\pi(j)}$, the algorithm returns the integral expression of $\mathbf{r}_{\pi(j)}$. In the algorithm, for any node $\mathbf{r}_{\pi(j)}$, $h_{\pi(j)}$ is the density function of the random variable $X_{\mathbf{r}_{\pi(j)}}$ and the bounds of the integral are θ_j^- and θ_j^+ which are given in Equation 5. Otherwise, it identifies the source nodes Src. A node is a source node if it has no incoming edge. The algorithm then constructs nested integrals corresponding to source nodes sequentially, and invokes the algorithm on the reduced graph obtained by deleting these source nodes new graph $\mathcal{G} \setminus$ Src. We discuss the impact of our optimization in the experiments presented in Table 2.

Example 3. Consider Example 2 on Page 6. For the SVT-Gauss program given in Algorithm 2, there is only one final state τ corresponding to the output $[0, 1]$ on input $[0, 1]$, where $\tau = (\alpha, \beta, G)$ in $\rho(u, o, \mathsf{P})$ where $\alpha(out_1) = 0$, $\alpha(out_2) = 1$ and $G = G_{\mathsf{rand}} = \{\mathbf{r}_1 < \mathbf{r}_T, \mathbf{r}_2 \ge \mathbf{r}_T\}$.

We can write $\mathsf{bpr}(\epsilon, \tau, \mathsf{th})$ as the expression

$$\mathcal{E} = \int_{-\mathsf{th}\cdot\frac{2}{\epsilon}}^{\mathsf{th}\cdot\frac{2}{\epsilon}} f_{0,\frac{2}{\epsilon}}(\mathbf{r}_T) \int_{-\mathsf{th}\cdot\frac{4}{\epsilon}}^{\mathbf{r}_T} f_{0,\frac{4}{\epsilon}}(\mathbf{r}_1) \int_{\mathbf{r}_T}^{1+\mathsf{th}\cdot\frac{4}{\epsilon}} f_{1,\frac{4}{\epsilon}}(\mathbf{r}_2)\, d\mathbf{r}_2\, d\mathbf{r}_1\, d\mathbf{r}_T.$$

[7] A weakly connected component in a directed graph is a maximal sub-graph such that the undirected version of the sub-graph, obtained by replacing each directed edge by an undirected edge, is connected

Figure 2 shows the dependency graph for $\mathcal{E}$. Finally, the optimization algorithm (Algorithm 5) rewrites $\mathcal{E}$ as

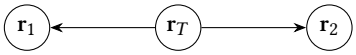

Figure 2: **Dependency graph of $\mathcal{E}$ in Example 3.**

$$\int_{-\mathsf{th}\cdot\frac{2}{\epsilon}}^{\mathsf{th}\cdot\frac{2}{\epsilon}} f_{0,\frac{2}{\epsilon}}(\mathbf{r}_T) \left(\int_{-\mathsf{th}\cdot\frac{4}{\epsilon}}^{\mathbf{r}_T} f_{0,\frac{4}{\epsilon}}(\mathbf{r}_1)\, d\mathbf{r}_1 \right) \left(\int_{\mathbf{r}_T}^{1+\mathsf{th}\cdot\frac{4}{\epsilon}} f_{1,\frac{4}{\epsilon}}(\mathbf{r}_2)\, d\mathbf{r}_2 \right) d\mathbf{r}_T.$$

7 Implementation and Evaluation

We implemented a simplified version of the algorithm, presented earlier, called the tool DiPApprox. The tool is built using Python and C++ and is designed to handle DiPGauss programs[8], determining whether they are differentially private, not differentially private, or Unknown. Given an input program P and an adjacency relation Φ, the tool checks differential privacy for fixed values of ϵ, ϵ_{prv} and δ.

DiPApprox uses three libraries: PLY [14] for program parsing, IGRAPH [24] for graph operations, and the FLINT library [50] for computing integral expressions. After parsing the program, we evaluate all final states of a program P (as given in Section 4). Afterwards, we represent each final state as a graph, perform the ordering of integrals and compute their limits as described in Section 6. Once such integral expressions are generated, we encode them in C++, and use FLINT to compute the interval probabilities of each final state for each input from the input pairs in the adjacency relation. Additionally, the tool refines the precision level to a higher level if the interval is too large to prove or disprove differential privacy. DiPApprox is available for download at [3].

7.1 Examples

In this section, we present a limited set of examples from our benchmark suite due to space constraints. Details of the remaining examples and additional experimental results are provided in Appendix F and Appendix G.

SVT Variants. We categorized SVT variants into three groups: SVT with Gaussian noise, SVT with Laplace noise, and SVT with mixed noise (where the threshold is sampled from a Laplace distribution and the queries from a Gaussian distribution, or vice versa). An example from the first category is SVT-Gauss (Algorithm 1).

Noisy-Max and Noisy-Min. In addition to the SVT variants, we examine Noisy-Max and Noisy-Min with Gaussian or Laplacian noise. The Noisy Max with Gaussian algorithm (see Appendix F.2) selects the query with the highest value after independently adding Gaussian noise to each query result. This approach obscures the true maximum, ensuring differential privacy [29, 33].

k-Min-Max and m-Range. The k-Min-Max algorithm (for $k \ge 2$) [19] perturbs the first k queries, computes the noisy maximum and minimum, and then checks whether each subsequent noisy query falls within this range; if not, the algorithm exits. The m-Range algorithm [19] perturbs $2m$ thresholds that define a rectangle of m dimensions and checks whether noisy queries lie within these

[8] Currently, DiPApprox only supports comparison amongst sampled values.

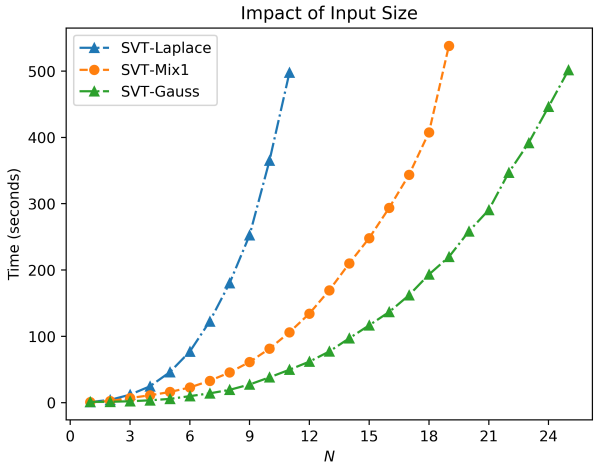

Figure 3: **Scaling behavior of SVT-Gauss, SVT-Laplace, and SVT-Mix1 with varying input size N for a single input pair.**

noisy limits. While the original algorithms use Laplace noise, we examine them also when the noise added is Gaussian.

7.2 Experiments

We evaluated DiPApprox on a macOS computer equipped with a 1.4 GHz Quad-Core Intel Core i5 processor and 8GB of RAM. Each example was executed three times, and the average execution time was recorded across these runs. Recall that when converting improper integrals with infinite limits into proper ones, we replace $-\infty$ and ∞ with tail bounds. We choose these bounds so that the remaining area in the tails is very small. We use $\mathrm{th} = 4$ for Gaussian distributions and $\mathrm{th} = 8$ for Laplace distributions while computing these tail bounds and the error. We used an initial precision of $\varrho = 16$ bits, which was refined up to 32 bits if differential privacy could not be verified. *All pairs* for input size N refers to all pairs of input vectors in $\{0,1\}^N$. Any pair of inputs in the $\{0,1\}^N$ is adjacent. A *single pair* of input size N refers to the pair of vector $(0^N, 0^{N-1}1)$, where the first vector consists of N zeros and the second vector consists of $N-1$ zeros followed by a one.

Performance and Scalability. We experimented on variants of SVT with input sizes from 1 to 26, on Noisy-Max and Noisy-Min with input sizes from 1 to 5, on m-Range with $m = 2$ and input sizes from 1 to 3, and on k-Min-Max with $k = 2$ and input sizes 3 and 4. DiPApprox can verify whether a SVT-Gauss variant is differentially private for a single input pair up to size $N = 25$. For larger inputs, it times out. Note that our timeout is set to 10 minutes; increasing this limit would allow DiPApprox to handle more examples. As N increases in the SVT variants, we encounter out-of-memory error (O.M) while storing probabilities for the case of all input pairs in the SVT variants as the number of input combinations grows exponentially with N. Performance results are presented in Table 1 and Figure 3 shows the scaling behavior with respect to input size N and runtime for a single input pair.

Example	N	Final States	$\overline{\lvert G \rvert}$	Avg. Depth	Single Pair DP?	Single Pair Time (s)	All Pairs DP?	All Pairs Time (s)
SVT-Gauss	2	3	1.7	2.3	✓	1.6	✓	2.4
	5	6	3.3	2.7	✓	7.9	✓	76.7
	25	26	13.5	2.9	✓	441.3	–	O.M
SVT-Gauss-Leaky-1	5	6	3.3	1.7	×	1.0	×	1.4
	6	7	3.9	1.7	×	1.0	×	2.1
SVT-Gauss-Leaky-2	3	4	2.2	1.5	×	1.0	×	1.0
	6	7	3.9	1.7	×	1.0	×	1.1
SVT-Mix1	2	3	1.7	2.3	✓	2.4	✓	4.3
	5	6	3.3	2.7	✓	19.5	✓	285.4
	17	18	9.4	2.9	✓	365.7	–	O.M
Noisy-Max-Gauss	2	2	1.0	2.0	✓	1.0	✓	1.3
	3	4	2.0	2.5	✓	1.6	✓	3.1
	4	8	3.0	3.0	✓	37.8	✓	303.0
Noisy-Min-Gauss	2	2	1.0	2.0	✓	1.0	✓	1.3
	3	4	2.0	2.5	✓	1.6	✓	3.1
	4	8	3.0	3.0	✓	36.8	✓	303.6
m-Range-Gauss	1	7	3.0	2.5	✓	1.5	✓	1.8
	2	13	4.2	3.2	✓	171.2	✓	344.4
	3	19	5.2	3.8	–	T.O	–	T.O
k-Min-Max-Gauss	3	16	4.0	3.0	✓	2.1	✓	5.7
	4	28	5.1	3.4	✓	41.2	✓	335.7

Table 1: **Summary of Experimental Results for DiPApprox. The columns in the table are defined as follows: N is the input size of the program. DP? indicates whether the program is differentially private. Final States denotes the number of final states. $\overline{\lvert G \rvert}$ and Avg. Depth, respectively, denote the average number of conditions and the average nesting depth of integral expressions, per final state. Time is the average time (in seconds) to verify differential privacy, measured over three runs. T.O indicates a timeout (exceeding 10 minutes), and O.M denotes a run out of memory. Differential privacy checks were performed with $\epsilon = 0.5$ and $\delta = 0.01$, except for SVT-Gauss-Leaky-1, which uses $\epsilon = 8$. We used $\epsilon_{\mathrm{prv}} = 0.5$, except for SVT-Gauss and SVT-Mix1, where $\epsilon_{\mathrm{prv}} = 1.24$.**

Impact of Optimization. We conducted experiments to evaluate the impact of the optimized integral ordering presented in Algorithm 5 on SVT-Gauss, SVT-Laplace, SVT-Mix1, and Noisy-Max-Gauss with input sizes ranging from $N = 1$ to $N = 5$. We compared the performance of the unoptimized version (which uses topological sorting for integral ordering) with the optimized version. The results indicate that the optimized approach leads to a significant reduction in the maximum integration depth for all examples. These improvements also translate into substantial reductions in the overall running time. Table 2 summarizes the optimization results.

7.3 Comparison with DiPC

We compare the performance of our tool, DiPApprox, with DiPC [4]. We chose DiPC for comparison as it allows for verification of approximate differential privacy (and not just pure differential privacy), i.e., it allows for verifying $(\epsilon_{\mathrm{prv}}, \delta)$ differential privacy for fixed values of $\epsilon, \epsilon_{\mathrm{prv}}$ and δ. Like DiPApprox, DiPC checks differential privacy for programs where both inputs and outputs are drawn from a finite domain and have bounded length. The key differences between DiPC and DiPApprox are as follows: (1) Unlike DiPApprox, DiPC does not support Gaussian distributions; (2) DiPC can check pure differential privacy for all values of $\epsilon > 0$ as well as for fixed values of ϵ. It also checks $(\epsilon_{\mathrm{prv}}, \delta)$ differential privacy for fixed values of $\epsilon, \epsilon_{\mathrm{prv}}$ and δ. DiPApprox, on the other hand can only check for

Example	N	Unoptimized			Optimized		
		Max Depth	Avg. Depth	Time (s)	Max Depth	Avg. Depth	Time (s)
SVT-Gauss	1	2	2.0	1.03	2	2.0	1.0
	2	3	2.67	2.27	3	2.3	1.6
	3	4	3.25	T.O	3	2.5	2.77
	4	5	3.8	T.O	3	2.6	3.48
	5	6	4.33	T.O	3	2.7	7.9
SVT-Mix1	1	2	2.0	1.01	2	2.0	0.94
	2	3	2.67	7.62	3	2.3	2.4
	3	4	3.25	T.O	3	2.5	7.88
	4	5	3.8	T.O	3	2.6	11.8
	5	6	4.33	T.O	3	2.7	19.5
Noisy-Max-Gauss	2	2	2.0	0.98	2	2.0	1.0
	3	3	3.0	3.53	3	2.5	1.6
	4	4	4.0	T.O	4	3.0	37.8
	5	5	5.0	T.O	5	3.5	T.O

Table 2: **Summary of optimization results for DiPApprox. The columns in the table are as follows: N represents the input size for the program. Time refers to the time taken to check differential privacy for a single pair, measured in seconds and averaged over three executions.** T.O **indicates a timeout (exceeding 10 minutes). Avg. Depth refers to the average nested depth of integrals across all executions. Max Depth refers to the maximum nested depth among all executions. Differential privacy checks were performed with $\epsilon = 0.5$ and $\delta = 0.01$. We used $\epsilon_{\text{prv}} = 1.24$ for SVT-Gauss and SVT-Mix1, and $\epsilon_{\text{prv}} = 0.5$ for Noisy-Max-Gauss.**

Example	N	ϵ	Time (s)		Speedup	DP?
			DiPC [4]	DiPApprox		
SVT-Laplace	1	0.5	25	1	25.0	✓
	2	0.5	106	32	3.31	✓
	3	0.5	578	279	2.07	✓
	4	0.5	2850	1638	1.74	✓
Noisy-Max-Laplace	3	1	278	166	1.67	✓
	3	0.5	311	152	2.05	✓
Noisy-Min-Laplace	3	1	180	165	1.09	✓
	3	0.5	286	154	1.86	✓
SVT-Laplace-Leaky-4	2	1	80	167	0.48	×
SVT-Laplace-Leaky-5	2	0.5	7	1	7.0	×
SVT-Laplace-Leaky-6	3	1	526	1075	0.49	×

Table 3: **Summary of comparison with** DiPC**. The table reports performance for both tools. The columns are as follows: N denotes the input size of the program. Time indicates the average time (in seconds) to verify differential privacy over three runs. DP? indicates whether the program is differentially private. Speedup represents the ratio of the time taken by** DiPC **to that of DiPApprox, indicating the relative performance gain. Differential privacy checks were performed with $\delta = 0$. For all examples in the table, $\epsilon_{\text{prv}} = \epsilon$.**

fixed values of ϵ_{prv} and δ. (3) DiPC relies on the proprietary software Wolfram Mathematica® for checking the encoded formulas, whereas DiPApprox uses the open-source library FLINT.

Our comparison is summarized in Table 3. DiPC takes more time to determine differential privacy for most examples. In some cases, DiPApprox significantly outperforms DiPC in verification time.

7.4 Discussion

Some salient insights from our experiments are as follows.

(1) DiPApprox can determine whether programs in DiPGauss are differentially private or not differentially private for several interesting examples. If a program is not differentially private, it provides a counterexample.
(2) DiPApprox demonstrates scalability, handling input sizes of up to 25 for some SVT variants in the case of a single input pair.
(3) The optimization algorithm significantly reduces the running time, achieving a substantial decrease, and lowers the nested depth of integral expressions.
(4) Verification involving the Laplace distribution takes longer than the Gaussian distribution. The Laplace distribution does not have a holomorphic extension to the complex numbers as it involves an absolute value term. This makes integration in FLINT more computationally intensive.

8 Related Work

Differential privacy was first introduced in [30]. For a comprehensive introduction to the topic, techniques, and results, consult the recent book [33] and survey [23]. Industry implementation of differential privacy include U.S. Census Bureau's LEHD OnTheMap tool [42], Google's RAPPOR system [34], Apple's DP implementation [27, 51], and Microsoft's Telemetry collection [28].

The subtle nature of the correctness proofs of differential privacy has prompted an interest in developing automated techniques to verify them. The main approaches to verification include the use of type systems [25, 26, 35, 45, 53, 55], probabilistic coupling [1, 7, 11, 13], using shadow executions [54], and simulation-based methods [52], and machine-checked proofs [48, 49].

Automated methods for verifying privacy include the use of hypothesis testing [29], symbolic differentiation [16], and program analysis [4, 19, 53]. Notably, [19, 53] even allow for verification with unbounded inputs, and for any $\epsilon > 0$. However, they do not allow sampling from Gaussians, and verify only pure differential privacy (i.e., $\delta = 0$).

Probabilistic model checking-based approaches are used in [20, 22, 40], where it is assumed that the program is given as a finite Markov Chain, ϵ is fixed to a concrete value. Sampling from continuous random variables is not allowed in [20, 22, 40]. Almost all the automated methods discussed so far are of checking ϵ-differential privacy; none work for (ϵ, δ)-differential privacy, except for [4].

The decision problem of checking (ϵ, δ)-differential privacy (and therefore also ϵ-differential privacy) was studied in [4] where the problem was shown to be undecidable in general, and a decidable sub-class of programs that sample from Laplacians was identified. Smaller classes of algorithms that use sampling from Laplacians and comparison of sampled values were identified in [18, 19] where it was shown that for them, the verification for *unbounded* inputs is decidable. The complexity of deciding differential privacy for randomized Boolean circuits and while programs is shown to be **coNP**$^{\#\mathbf{P}}$-complete and **PSPACE**-complete in [36] and [17] respectively when the number of inputs is finite, probabilistic choices are fair coin tosses and e^{ϵ} is a rational number.

None of the automated verification approaches discussed above verify differential privacy of programs that sample from Gaussians.

9 Conclusions and Future Work

We addressed the problem of verifying differential privacy for parameterized programs that support sampling from Gaussian and Laplace distributions, and operate over finite input and output domains. For a class of loop-free programs called DiPGauss, we

showed that the problem of determining if a given program P, with privacy parameter ϵ, is $(\epsilon_{\mathsf{prv}}, \delta)$-differential privacy for given rational values $\epsilon > 0$, $\epsilon_{\mathsf{prv}} > 0$, and $\delta \in [0, 1]$ is almost decidable: it is decidable for all values of δ in $[0, 1]$, except for a finite set of exceptional values determined by P, ϵ, ϵ_{prv}, and the adjacency relation. We establish this through an algorithm, $\mathsf{VerifyDP}_{\varrho}$, which outputs one of three results: DP, Not_DP, or Unknown. Our implementation of $\mathsf{VerifyDP}_{\varrho}$ leverages the FLINT library for evaluating definite integrals and incorporates several performance optimizations, such as reducing the nesting depth of integrals to enhance scalability on practical benchmarks. The algorithm is implemented in our tool DiPApprox and has been empirically evaluated on a variety of examples drawn from the literature.

For future work, it would be interesting to explore extensions in three directions: (i) allowing unbounded loops as well as nonlinear functions of real variables in programs, (ii) generalizing input and output domains to real or rational values, and (iii) enabling the privacy parameter ϵ to range over an interval, with ϵ_{prv} (or δ) specified as a function of ϵ (or ϵ_{prv}, respectively).

Acknowledgements. This work was partially supported by the National Science Foundation: Bishnu Bhusal and Rohit Chadha were partially supported by grant CCF 1900924, A. Prasad Sistla was partially supported by grant CCF 1901069, and Mahesh Viswanathan was partially supported by grants CCF 1901069 and CCF 2007428.

References

[1] Aws Albarghouti and Justin Hsu. 2018. Synthesizing coupling proofs of differential privacy. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*. 58:1–58:30.

[2] Kareem Amin, Alex Bie, Weiwei Kong, Alexey Kurakin, Natalia Ponomareva, Umar Syed, Andreas Terzis, and Sergei Vassilvitskii. 2024. Private prediction for large-scale synthetic text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 7244–7262. doi:10.18653/v1/2024.findings-emnlp.425

[3] Anonymous. 2025. Artifact for Approximate Algorithms for Verifying Differential Privacy with Gaussian Distributions. https://anonymous.4open.science/r/approximate-dp-CA17/README.md.

[4] G. Barthe, R. Chadha, V. Jagannath, A.P. Sistla, and M. Viswanathan. 2020. Deciding Differential Privacy for Programs with finite inputs and outputs. In *Proceedings of the IEEE Symposium on Logic in Computer Science*. 141–154.

[5] G. Barthe, R. Chadha, P. Krogmeier, A.P. Sistla, and M. Viswanathan. 2021. Deciding accuracy of differentially private schemes. In *Proceedings of the ACM Symposium on the Principles of Programming Languages*. 1–30.

[6] G. Barthe, G.P. Farina, M. Gaboardi, E.J.G. Arias, A. Gordon, J. Hsu, and P.-Y. Strub. 2016. Differentially private Bayesian programming. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 68–79.

[7] Gilles Barthe, Noémie Fong, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2016. Advanced Probabilistic Couplings for Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 55–67.

[8] G. Barthe, M. Gaboardi, E.J. Arias, J. Hsu, C. Kunz, and P. Strub. 2014. Proving Differential Privacy in Hoare Logic. In *Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium (CSF14)*. IEEE Computer Society, 266–277.

[9] G. Barthe, M. Gaboardi, E.J.G. Arias, J. Hsu, A. Roth, and P.-Y. Strub. 2015. Higher-order approximate relational refinement types for mechanism design and differential privacy. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 55–68.

[10] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. 2016. A program logic for union bounds. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. 107:1–1–7:15.

[11] G. Barthe, M. Gaboardi, B. Grégoire, J. Hsu, and P. Strub. 2016. Proving differential privacy via probabilistic couplings. In *Proceedings of the IEEE Symposium on Logic in Computer Science*. 749–758.

[12] G. Barthe, B. Köpf, F. Olmedo, and S.Z. Béguelin. 2012. Probabilistic reasoning for differential privacy. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 97–110.

[13] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella-Béguelin. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Transactions on Programming Languages and Systems* 35, 3 (2013), 9.

[14] David Beazley. 2022. GitHub - dabeaz/ply: Python Lex-Yacc — github.com. https://github.com/dabeaz/ply. [Accessed 24-Jan-2023].

[15] Daniel Berend and Luba Bromberg. 2006. Uniform decompositions of polytopes. *Applicationes Mathematicae* 33 (01 2006), 243–252. doi:10.4064/am33-2-7

[16] Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin T. Vechev. 2018. DP-Finder: Finding Differential Privacy Violations by Sampling and Optimization. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 508–524.

[17] M. Bun, M. Gaboardi, and L. Glinskih. 2022. The Complexity of Verifying Boolean Programs as Differentially Private. In *2022 IEEE 35th Computer Security Foundations Symposium (CSF)*. 396–411. doi:10.1109/CSF54842.2022.9919653

[18] R. Chadha, A.P. Sistla, and M. Viswanathan. 2021. On Linear time decidability of differential privacy for programs with unbounded inputs. In *Proceedings of the IEEE Symposium on Logic in Computer Science*. 1–13.

[19] R. Chadha, A.P. Sistla, M. Viswanathan, and B. Bhusal. 2023. Deciding Differential Privacy of Online Algorithms with Multiple Variables. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1761–1775.

[20] Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. 2014. Generalized Bisimulation Metrics. In *35th International Conference on Concurrency Theory (CONCUR)*. 32–46.

[21] Yan Chen and Ashwin Machanavajjhala. 2015. On the Privacy Properties of Variants on the Sparse Vector Technique. *CoRR* abs/1508.07306 (2015).

[22] D. Chistikov, S. Kiefer, A. S. Murawski, and D. Purser. 2020. The Big-O Problem for Labelled Markov Chains and Weighted Automata. In *31st International Conference on Concurrency Theory (CONCUR) (LIPIcs, Vol. 171)*. 41:1–41:19.

[23] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang. 2018. Privacy at Scale: Local Differential Privacy in Practice. In *Proceedings of the International Conference on Management of Data*. 1655–1658.

[24] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal* Complex Systems (2006), 1695. https://igraph.org

[25] Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. 2014. Really Natural Linear Indexed Type Checking. In *26th 2014 International Symposium on Implementation and Application of Functional Languages (IFL)*. 5:1–5:12.

[26] Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. 2019. Probabilistic Relational Reasoning via Metrics. In *Proceedings of the IEEE Symposium on Logic in Computer Science*. 1–19.

[27] Apple Differential Privacy Team. 2017. Learning with privacy at scale.

[28] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting Telemetry Data Privately. In *Advances in Neural Information Processing Systems 30*. 3571–3580.

[29] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2018. Detecting Violations of Differential Privacy. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 475–489.

[30] C. Dwork, F. McSherry, K. Nissim, and A. Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the International Conference on Theory of Cryptography*. 265–284.

[31] C. Dwork, M. Naor, O. Reingold, G. Rothblum, and S. Vadhan. 2009. On the complexity of differentially private data release: Efficient algorithms and hardness results. In *Proceedings of the ACM Symposium on Theory of Computation*. 381–390.

[32] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. 2009. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 381–390.

[33] Cynthia Dwork and Aron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, Vol. 9. Springer. 211–407 pages.

[34] U. Erlingsson, V. Pihur, and A. Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the ACM SIGSAC conference on computer and communications security*. 1054–1067.

[35] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B.C. Pierce. 2013. Linear Dependent Types for Differential Privacy. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 357–370.

[36] Marco Gaboardi, Kobbi Nissim, and David Purser. 2020. The Complexity of Verifying Loop-Free Programs as Differentially Private. In *47th International Colloquium on Automata, Languages, and Programming, (ICALP) (LIPIcs, Vol. 168)*. 129:1–129:17.

[37] A. Gupta, A. Roth, and J. Ullman. 2012. Iterative constructions and provate data release. In *Proceedings of the International Conference on Theory of Cryptography*. 339–356.

[38] M. Hardt and G.N. Rothblum. 2010. A multiplicative weights mechanism for privacy-preserving data analysis. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*. 61–70.

[39] K.-I. Ko. 1991. *Complexity Theory of Real Functions*. Birkhauser.

[40] Depeng Liu, Bow-Yaw Wang, and Lijun Zhang. 2018. Model Checking Differentially Private Properties. In *Programming Languages and Systems - 16th Asian Symposium, (APLAS) (Lecture Notes in Computer Science, Vol. 11275)*. 394–414.

[41] M. Lyu, D. Su, and N. Li. 2017. Understanding the Sparse Vector technique for differential privacy. *Proceedings of VLDB* 10, 6 (2017), 637–648.

[42] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. 2008. Privacy: From theory to practice on the map. In *Proceedings of the IEEE International Conference on Data Engineering*. 277–286.

[43] F. McSherry. 2009. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 19–30.

[44] P. Mohan, A. Thakurta, E. Shi, D. Song, and E. Culler. 2012. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

[45] J. Reed and B.C. Pierce. 2010. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming*. 157–168.

[46] Cody Rivera, Bishnu Bhusal, Rohit Chadha, A. Prasad Sistla, and Mahesh Viswanathan. 2025. Checking δ-Satisfiability of Reals with Integrals. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 105 (April 2025), 26 pages. doi:10.1145/3720446

[47] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. 2010. Airavat: Security and privacy for MapReduce. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*.

[48] Tetsuya Sato and Yasuhiko Minamide. 2025. Differential Privacy. *Archive of Formal Proofs* (January 2025). https://isa-afp.org/entries/Differential_Privacy.html, Formal proof development.

[49] Tetsuya Sato and Yasuhiko Minamide. 2025. Formalization of Differential Privacy in Isabelle/HOL. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 67–82.

[50] The FLINT team. 2023. *FLINT: Fast Library for Number Theory*. Version 3.0.0, https://flintlib.org.

[51] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudiger, V.R. Sridhar, and D. Davidson. 2017. Learning new words. US Patent 9,594,741.

[52] Michael Carl Tschantz, Dilsun Kirli Kaynar, and Anupam Datta. 2011. Formal Verification of Differential Privacy for Interactive Systems (Extended Abstract). In *27th Conference on the Mathematical Foundations of Programming Semantics (MFPS) (Electronic Notes in Theoretical Computer Science, Vol. 276)*. 61–79.

[53] Yuxin Wang, Zeyu Ding, Daniel Kifer, and Danfeng Zhang. 2020. CheckDP: An Automated and Integrated Approach for Proving Differential Privacy or Finding Precise Counterexamples. In *2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 919–938.

[54] Yuxin Wang, Zeyu Ding, Guanhong Wang, Daniel Kifer, and Danfeng Zhang. 2019. Proving differential privacy with shadow execution. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, (PLDI)*. 655–669.

[55] D. Zhang and D. Kifer. 2017. LightDP: Towards Automating Differential Privacy Proofs. In *44th ACM Symposium on Principles of Programming Languages (POPL17)*. Association for Computing Machinery, 266–277.

[56] Yuqing Zhu and Yu-Xiang Wang. 2020. Improving sparse vector technique with renyi differential privacy. *Advances in neural information processing systems* 33 (2020), 20249–20258.

A Proof of Lemma 2

($\Rightarrow$) Let P be $(\epsilon_{\text{prv}}, \delta)$ differentially private and $u, u' \in \Phi$. By setting $F = \{o \in \mathcal{V} \mid \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}] > 0\}$, we can conclude that Equation (4) is true for $\epsilon, \text{P}, u, u'$ as follows.

$$\begin{aligned}&\sum_{o\in\mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)\\ &= \sum_{o\in\mathcal{F}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)\\ &+ \sum_{o\in\mathcal{V}\setminus\mathcal{F}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)\end{aligned}$$

For $o \in F$, we have that

$$\begin{aligned}&\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)\\ &= \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}].\end{aligned}$$

For $o \notin F$, we have that

$$\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0) = 0$$

.

Thus, $\sum_{o\in\mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0) = \sum_{o\in\mathcal{F}} \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}] \leq \delta$ as P is $(\epsilon_{\text{prv}}, \delta)$ differentially private.

($\Leftarrow$) This direction follows from the observation that for each u, u' and arbitrary $F \subseteq \mathcal{V}$,

$$\begin{aligned}&\sum_{o\in F} \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}} \sum_{o\in F} \text{Prob}[\epsilon, u', o, \text{P}]\\ &= \sum_{o\in F}(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}])\\ &\leq \sum_{o\in F} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0)\\ &\leq \sum_{o\in\mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0).\end{aligned}$$

The second last line of the above sequence of inequalities follow from the fact that for any real number a, $a \leq \max(a, b)$. The last line follows from the fact that for all $o \notin F$, $\max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0) \geq 0$.

B Proofs of Lemma 4 and Lemma 5

We prove both Lemma 4 and Lemma 5 together. Let $(u, u') \in \Phi$. Let

$$\delta_{u,u'} = \sum_{o\in\mathcal{V}} \max(\text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}], 0).$$

Let $o \in \mathcal{V}$ be an output. After $\text{VerifyDP}_{\varrho}$ processes (u, u') and the output o, let $L_1(u, o), U_1(u, o), L_2(u', o)$, and $U_2(u', o)$ be such that values $\text{store}(u, o) = [L_1(u, o), U_2(u, o)]$ and $\text{store\_scale}[u', o] = [L_2(u', o), U_2(u', o)]$.

We have the following equations:

$$\begin{aligned}&L_1(u, o) \leq \text{Prob}[\epsilon, u, o, \text{P}] \leq L_1(u, o) + \frac{1}{2^{\varrho}}\\ &U_1(u, o) \geq \text{Prob}[\epsilon, u, o, \text{P}] \geq U_1(u, o) - \frac{1}{2^{\varrho}}\\ &L_2(u', o) \leq e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}] \leq L_2(u, o) + \frac{1}{2^{\varrho}}\\ &U_2(u', o) \geq e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}] \geq U_2(u, o) - \frac{1}{2^{\varrho}}.\end{aligned}$$

This implies that

$$\begin{aligned}L_1(u, o) - U_2(u', o) &\leq \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}]\\ &\leq L_1(u, o) - U_2(u', o) + \frac{2}{2^{\varrho}}\end{aligned}$$

and

$$\begin{aligned}U_1(u, o) - L_2(u', o) &\geq \text{Prob}[\epsilon, u, o, \text{P}] - e^{\epsilon_{\text{prv}}}\text{Prob}[\epsilon, u', o, \text{P}]\\ &\geq U_1(u, o) - L_2(u', o) - \frac{2}{2^{\varrho}}.\end{aligned}$$

Therefore, when $\text{VerifyPair}(u, u', \delta, \text{store}, \text{store\_scale})$ is executed then at the end of the for loop in VerifyPair, the following equations hold.

$$\Delta_{min} \leq \delta_{u,u'} \leq \Delta_{min} + \frac{2|\mathcal{V}|}{2^{\varrho}} \tag{6}$$

$$\Delta_{max} \geq \delta_{u,u'} \geq \Delta_{max} - \frac{2|\mathcal{V}|}{2^{\varrho}}. \tag{7}$$

Finishing Lemma 4 Proof. It is easy to see that Equation 6 and Equation 7 imply the two parts of the of Lemma 4.

Finishing Lemma 5 Proof. Recall that if output probabilities are effectively approximable, then so are scaled output probabilities. (Proposition 3). For part one of the Lemma 5, observe that Equation 6 implies that

$$\Delta_{min} \geq \delta_{u,u'} - \frac{2|\mathcal{V}|}{2^{\varrho}}$$

and thus

$$\Delta_{min} - \delta \geq (\delta_{u,u'} - \delta) - \frac{2|\mathcal{V}|}{2^{\varrho}}.$$

Now, part one of the Lemma 5 follows from the observation that if P is not differentially private then there must be $(u, u') \in \Phi$ such that $\delta_{u,u'} - \delta > 0$ and hence there is a precision ϱ_0 such $(\delta_{u,u'} - \delta) - \frac{2|\mathcal{V}|}{2^{\varrho_0}} > 0$.

For part two of the Lemma 5, observe that we have from Equation 7 that for each $(u, u') \in \Phi$,

$$-\Delta_{max} + \frac{2|\mathcal{V}|}{2^{\varrho}} \geq -\delta_{u,u'}.$$

Hence,

$$\delta - \Delta_{max} \geq (\delta - \delta_{u,u'}) - \frac{2|\mathcal{V}|}{2^{\varrho}}.$$

Now if P is differentially private and $\delta \notin \text{D}_{\text{P},\epsilon,\epsilon_{\text{prv}},\Phi}$, then $\delta - \delta_{u,u'} > 0$ for each $(u, u') \in \Phi$. As there are only a *finite* number of pairs $(u, u') \in \Phi$, it implies that

$$\min_{(u,u')\in\Phi} (\delta - \delta_{u,u'}) > 0.$$

From this, it is easy to see that there is a precision ϱ_0 such that $(\delta - \delta_{u,u'}) - \frac{2|\mathcal{V}|}{2^{\varrho_0}} > 0$ for each $(u, u') \in \Phi$. Thus, VerifyPair will return DP for each $(u, u') \in \Phi$ when run with precision ϱ_0.

C Proof of Lemma 6

Given a threshold th, it is easy to see that $0 \leq \text{tpr}(\epsilon, \tau, \text{th}) \leq \sum_{i=0}^{n-1} \text{tl}(\text{th}, X_{\mathbf{r}_i}, \mu_i, \sigma_i)$. From the known tail bounds (See Section 2), we know that that, for each i, there is a monotonically decreasing function $k_i(\cdot)$ such that $\text{tl}(\text{th}, X_{\mathbf{r}_i}, \mu_i, \sigma_i) \leq k_i(\text{th})$. Furthermore, $\lim_{\text{th}\to\infty} k_i(\text{th}) = 0$. From these observations and the known tail bounds, we can choose th_i such that $0 \leq \text{tl}(\text{th}, X_{\mathbf{r}_i}, \mu_i, \sigma_i) \leq \frac{1}{2n2^{\varrho}}$ for each $\text{th} \geq \text{th}_i$. The result follows by choosing $\text{th} = \max_{0\leq i\leq n-1} \text{th}_i$.

D Proof of Theorem 8

Thanks to Lemma 6, it suffices to show that we can compute a ϱ + 1-approximation $[L, U]$ of $\mathsf{bpr}(\epsilon, \tau, \mathsf{th})$. From our previous arguments, it follows that $\mathsf{bpr}(\epsilon, \tau, \mathsf{th})$ can be obtained as a finite sum of normalized integral expressions of the form $\mathcal{E}$ shown above. All the constants and coefficients of the linear functions used as limits of integrals in the different summands can be obtained algorithmically from the values of $\mathsf{th}, \epsilon, u, o, \mathsf{P}$. The result now follows from the fact that all the probability density functions in DiPGauss are computable, and that the set of computable functions are closed under summation, definite proper integration, and composition. [39, 46].

E Correctness of Algorithm 5

Algorithm 5 is given by a recursive function GenExpr($\mathcal{G}$), which takes the dependency graph $\mathcal{G}$ of an integral expression E in normalized form as an argument and outputs an integral expression equivalent to E, i.e., has the same value as E for any given values to the free variables in E; note E will not have free variables during the first invocation of GenExpr(), but in the subsequent recursive invocations the expression E (corresponding to the argument $\mathcal{G}$), may have free variables which appear in the limits of the integrals appearing in E.

Each node in $\mathcal{G}$ represents a random variable and an edge (u, v) in $\mathcal{G}$ indicates that the variable u appears in the lower or upper limit of the integral corresponding to the variable v.

If $\mathcal{G}$ has more than one weakly connected component, say $\mathcal{G}$ has two such components $\mathcal{G}_1, \mathcal{G}_2$. The integrals corresponding to the variables in $\mathcal{G}_1$ can be moved leftwards to the front retaining their order of occurrence in E, resulting in an expression which is a product of two expressions E_1 (corresponding to $\mathcal{G}_1$) and E_2 (corresponding to $\mathcal{G}_2$). The resulting product expression is equivalent to E since none of the variables in G_1 depend on those in $\mathcal{G}_2$. This argument generalizes when $\mathcal{G}$ has more than two weakly connected components. This argument holds for the recursive invocation GenExpr($\mathcal{G}'$) in the return statement of the function.

F Pseudocode of Examples

We present a short description and pseudo-code of the examples from our benchmark suite.

F.1 SVT variants

We have the following variants: SVT-Gauss, SVT-Laplace, SVT-Mix1, and SVT-Mix2. These are similar algorithms that differ only in the distributions from which they sample noise. SVT-Gauss samples both the threshold and the queries from a Gaussian distribution. SVT-Laplace samples both from a Laplace distribution. SVT-Mix1 samples the threshold from a Gaussian distribution and the queries from a Laplace distribution, while SVT-Mix2 does the opposite. These algorithms output $\top$ when the noisy query result is less than or equal to the noisy threshold; otherwise, they output $\bot$. We also have non-private variants of SVT-Gauss: SVT-Gauss-Leaky-1 and SVT-Gauss-Leaky-2. SVT-Gauss-Leaky-1 compares noisy queries with a non-noisy threshold, whereas SVT-Gauss-Leaky-2 compares a noisy threshold with non-noisy queries. Additionally, we consider four non-private variants of SVT-Laplace, borrowed from [4]: SVT-Laplace-Leaky-3, SVT-Laplace-Leaky-4, SVT-Laplace-Leaky-5, and SVT-Laplace-Leaky-6.

Another set of examples of SVT variants includes SVT-Gauss-Ge, SVT-Laplace-Ge, SVT-Mix1-Ge, and SVT-Mix2-Ge. These algorithms are also distinguished by the distributions from which they sample noise. SVT-Gauss-Ge samples both the threshold and the queries from a Gaussian distribution. SVT-Laplace-Ge samples both from a Laplace distribution. SVT-Mix1-Ge samples the threshold from a Gaussian distribution and the queries from a Laplace distribution, while SVT-Mix2-Ge does the opposite. These algorithms output $\top$ when the noisy query result is greater than or equal to the noisy threshold; otherwise, they output $\bot$. We also have non-private versions of SVT-Gauss-Ge: SVT-Gauss-Ge-Leaky-1 and SVT-Gauss-Ge-Leaky-2. SVT-Gauss-Ge-Leaky-1 compares noisy queries with a non-noisy threshold, whereas SVT-Gauss-Ge-Leaky-2 compares a noisy threshold with non-noisy queries.

Algorithm 6: SVT-Gauss

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 7: SVT-Laplace

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 8: SVT-Mix1

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 9: SVT-Mix2

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 10: SVT-Gauss-Leaky-1

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow T$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{2\Delta}{\epsilon})$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 11: SVT-Gauss-Leaky-2

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow q[i]$
  if $\mathbf{r} \geq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 12: SVT-Laplace-Leaky-3

Input: $q[1:N]$
Output: $out[1:N]$
$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{2c\Delta}{\epsilon})$
  b $\leftarrow \mathbf{r} \geq \mathbf{r}_T$
  if b **then**
    $out[i] \leftarrow \mathsf{Disc}_{\mathsf{seq}}(\mathbf{r})$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 13: SVT-Laplace-Leaky-4

Input: $q[1:N]$
Output: $out[1:N]$
$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{4\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{4\Delta}{3\epsilon})$
  b $\leftarrow \mathbf{r} \geq \mathbf{r}_T$
  if b **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 14: SVT-Laplace-Leaky-5

Input: $q[1:N]$
Output: $out[1:N]$
$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow q[i]$
  $\mathrm{b} \leftarrow \mathbf{r} \geq \mathbf{r}_T$
  if b **then**
    $out[i] \leftarrow \top$
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 15: SVT-Laplace-Leaky-6

Input: $q[1:N]$
Output: $out[1:N]$
$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{2\Delta}{\epsilon})$
  $\mathrm{b} \leftarrow \mathbf{r} \geq \mathbf{r}_T$
  if b **then**
    $out[i] \leftarrow \top$
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 16: SVT-Gauss-Ge

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

F.2 Noisy-Min and Noisy-Max

We have four variants of Noisy-Min and Noisy-Max: Noisy-Min-Gauss, Noisy-Min-Laplace, Noisy-Max-Gauss, and Noisy-Max-Laplace. Noisy-Min-Gauss and Noisy-Min-Laplace are similar algorithms that differ only in the noise distribution: Noisy-Min-Gauss uses Gaussian noise, whereas Noisy-Min-Laplace uses Laplace noise. These algorithms add noise to each query and perform an *argmin* operation, returning the index of the noisy minimum value.

Algorithm 17: SVT-Laplace-Ge

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 18: SVT-Mix1-Ge

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathsf{Lap}(T, \frac{2\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 19: SVT-Mix2-Ge

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathsf{Lap}(q[i], \frac{4\Delta}{\epsilon})$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    $count \leftarrow count + 1$
    if $count \geq c$ **then**
      exit
    end
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 20: SVT-Gauss-Ge-Leaky-1

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow T$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{2\Delta}{\epsilon})$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Algorithm 21: SVT-Gauss-Ge-Leaky-2

Input: $q[1:N]$
Output: $out[1:N]$

$\mathbf{r}_T \leftarrow \mathcal{N}(T, \frac{2\Delta}{\epsilon})$
for $i \leftarrow 1$ **to** N **do**
  $\mathbf{r} \leftarrow q[i]$
  if $\mathbf{r} \leq \mathbf{r}_T$ **then**
    $out[i] \leftarrow \top$
    exit
  else
    $out[i] \leftarrow \bot$
  end
end

Similarly, Noisy-Max-Gauss and Noisy-Max-Laplace are also similar algorithms that differ only in the noise distribution: Noisy-Max-Gauss uses Gaussian noise, whereas Noisy-Max-Laplace uses Laplace noise. These algorithms add noise to each query and perform an *argmax* operation, returning the index of the noisy maximum value.

Algorithm 22: Noisy-Max-Gauss

Input: $q[1:N]$
Output: out

NoisyVector ← []
for $i \leftarrow 1$ **to** N **do**
  NoisyVector[i] ← $\mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
end
out ← argmax(NoisyVector)

F.3 k-Min-Max and m-Range

The k-Min-Max algorithm (for $k \geq 2$) perturbs the first k queries with Laplace noise, computes the noisy maximum and minimum, and then checks whether each subsequent noisy query falls within this range; if not, the algorithm exits. The m-Range algorithm perturbs $2m$ thresholds that define a rectangle of m dimensions and checks whether noisy queries lie within these noisy limits.

Algorithm 23: Noisy-Min-Gauss

Input: $q[1:N]$
Output: out

NoisyVector ← []
for $i \leftarrow 1$ **to** N **do**
  NoisyVector[i] ← $\mathcal{N}(q[i], \frac{4\Delta}{\epsilon})$
end
out ← argmin(NoisyVector)

Algorithm 24: Noisy-Max-Laplace

Input: $q[1:N]$
Output: out

NoisyVector ← []
for $i \leftarrow 1$ **to** N **do**
  NoisyVector[i] ← Lap($q[i]$, $\frac{2}{\epsilon}$)
end
out ← argmax(NoisyVector)

Algorithm 25: Noisy-Min-Laplace

Input: $q[1:N]$
Output: out

NoisyVector ← []
for $i \leftarrow 1$ **to** N **do**
  NoisyVector[i] ← Lap($q[i]$, $\frac{2}{\epsilon}$)
end
out ← argmin(NoisyVector)

G Full Experimental Results

Here, we present the complete experimental results. Table 7 shows the performance results of our benchmark suite. Table 6 illustrates the impact of optimization on SVT-Gauss, SVT-Laplace, SVT-Mix1, and Noisy-Max-Gauss. Table 5 provides a comparison with the DiPC tool. Table 4 demonstrates the effect of varying ϵ and δ. Section G.1 discusses the comparison with CheckDP [53].

G.1 Comparison with CheckDP

We have compared our tool with CheckDP [53]. However, our tool verifies privacy only for fixed values of ϵ, whereas CheckDP verifies for all values of $\epsilon > 0$. Additionally, our tool supports checking of (ϵ, δ)-differential privacy and can handle programs with Gaussian distributions, whereas CheckDP only supports ϵ-differential privacy and programs with Laplace distributions. Table 8 presents the results of the comparison.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

ϵ \ δ	0.01	0.05	0.10	0.20	0.30	0.40	0.50	0.60
0.05	7.13	6.37	6.59	6.24	6.38	8.35	7.95	6.68
0.08	6.71	7.62	8.23	6.82	7.35	7.59	7.49	7.58
0.09	7.35	7.05	6.51	6.45	7.00	9.25	9.76	9.49
0.10	6.80	6.40	6.45	6.50	6.24	6.09	6.12	6.01
0.20	6.85	6.17	5.88	5.96	6.05	6.35	5.96	5.83
0.30	5.89	5.84	5.81	6.27	6.05	5.82	5.93	5.78
0.40	5.80	6.35	6.11	5.99	6.01	6.08	5.99	5.96
0.50	5.62	5.57	5.68	5.58	5.55	5.76	5.66	5.54
0.60	6.06	7.01	6.53	6.07	6.06	5.98	6.14	6.05
0.70	8.57	6.95	6.72	6.38	5.99	6.47	6.90	6.07
0.80	6.04	5.98	5.90	6.01	6.06	6.00	5.93	5.97
0.90	5.89	5.97	5.96	5.93	5.95	6.02	5.98	6.14
1.00	5.64	5.68	5.70	5.65	5.67	5.66	5.63	5.72

Table 4: **Summary of the impact of varying ϵ and δ on the SVT-Gauss-Ge example with an input size of $N = 5$. In all cases, we used $\epsilon_{\text{prv}} = \epsilon$.**

Example	N	ϵ	Time		Speedup	DP?
			DiPC [4]	DiPApprox		
SVT-Laplace	1	1	52	1	52.0	✓
	1	0.5	25	1	25.0	✓
	2	1	104	26	4.0	✓
	2	0.5	106	32	3.31	✓
	3	1	558	250	2.23	✓
	3	0.5	578	279	2.07	✓
	4	1	2814	1481	1.9	✓
	4	0.5	2850	1638	1.74	✓
SVT-Laplace-Ge	1	1	29	1	29.0	✓
	1	0.5	23	1	23.0	✓
	2	1	145	25	5.8	✓
	2	0.5	163	22	7.41	✓
	3	1	906	227	3.99	✓
	3	0.5	1134	204	5.56	✓
	4	1	4317	1684	2.56	✓
	4	0.5	4887	1285	3.8	✓
Noisy-Max-Laplace	3	1	278	166	1.67	✓
	3	0.5	311	152	2.05	✓
Noisy-Min-Laplace	3	1	180	165	1.09	✓
	3	0.5	286	154	1.86	✓
SVT-Laplace-Leaky-4	2	1	80	167	0.48	×
SVT-Laplace-Leaky-5	2	0.5	7	1	7.0	×
SVT-Laplace-Leaky-6	3	1	526	1075	0.49	×

Table 5: **Summary of comparison with DiPC. The table reports performance for both tools. The columns are as follows: N denotes the input size of the program. Time indicates the average time (in seconds) to verify differential privacy over three runs. DP? indicates whether the program is differentially private. Speedup represents the ratio of the time taken by DiPC to that of DiPApprox, indicating the relative performance gain. Differential privacy checks were performed with $\delta = 0$. For all examples in the table, $\epsilon_{\text{prv}} = \epsilon$.**

Example	N	Unoptimized			Optimized		
		Max Depth	Avg. Depth	Time	Max Depth	Avg. Depth	Time
SVT-Gauss	1	2	2.0	1.03	2	2.0	1.0
	2	3	2.67	2.27	3	2.3	1.6
	3	4	3.25	T.O	3	2.5	2.77
	4	5	3.8	T.O	3	2.6	3.48
	5	6	4.33	T.O	3	2.7	7.9
SVT-Laplace	1	2	2.0	1.04	2	2.0	1.0
	2	3	2.67	14.21	3	2.3	5.1
	3	4	3.25	T.O	3	2.5	12.79
	4	5	3.8	T.O	3	2.6	26.2
	5	6	4.33	T.O	3	2.7	47.6
SVT-Mix1	1	2	2.0	1.01	2	2.0	0.94
	2	3	2.67	7.62	3	2.3	2.4
	3	4	3.25	T.O	3	2.5	7.88
	4	5	3.8	T.O	3	2.6	11.8
	5	6	4.33	T.O	3	2.7	19.5
Noisy-Max-Gauss	2	2	2.0	0.98	2	2.0	1.0
	3	3	3.0	3.53	3	2.5	1.6
	4	4	4.0	T.O	4	3.0	37.8
	5	5	5.0	T.O	5	3.5	T.O

Table 6: **Summary of optimization results for DiPApprox. The columns in the table are as follows: N represents the input size for the program. Time refers to the time taken to check differential privacy for a single pair, measured in seconds and averaged over three executions. T.O indicates a timeout (exceeding 10 minutes). Avg. Depth refers to the average nested depth of integrals across all executions. Max Depth refers to the maximum nested depth among all executions. The optimized algorithm corresponds to Algorithm 5. Differential privacy checks were performed with $\epsilon = 0.5$ and $\delta = 0.01$.**

Algorithm 26: m-Range-Gauss

Input: $q[1:m]$
Output: $out[1:Nm]$

for $j \leftarrow 1$ **to** m **do**
  low[j] $\leftarrow$ Lap($T_1[j]$, $\frac{4m}{\epsilon}$)
  high[j] $\leftarrow$ Lap($T_2[j]$, $\frac{4m}{\epsilon}$)
  $out[j] \leftarrow$ cont
end
for $i \leftarrow 1$ **to** N **do**
  for $j \leftarrow 1$ **to** m **do**
    r $\leftarrow$ Lap($q[m(i-1)+j]$, $\frac{4}{\epsilon}$)
    if (r $\geq$ low[j]) $\wedge$ (r $<$ high[j]) **then**
      $out[m(i-1)+j] \leftarrow$ cont
    else if ((r $\geq$ low[j]) $\wedge$ (r $>$ high[j])) **then**
      $out[m(i-1)+j] \leftarrow \top$
      exit
    end
    else if ((r $<$ low[j]) $\wedge$ (r $<$ high[j])) **then**
      $out[m(i-1)+j] \leftarrow \bot$
      exit
    end
  end
end

Example	N	Time	
		CheckDP	DiPApprox
SVT-Gauss	1	29.9	1.3
m-Range-Laplace	1	T.O	19.3
k-Min-Max-Laplace	3	T.O	200.3

Table 8: **Summary of comparison with** CheckDP**. The table reports performance for both tools. The columns are as follows: N denotes the input size of the program. Time indicates the average time (in seconds) to verify differential privacy over three runs. Differential privacy checks were performed with $\epsilon_{\text{prv}} = \epsilon = 0.5$ and $\delta = 0$ for DiPApprox.**

Example	N	Final States	$\overline{\lvert G \rvert}$	Avg. Depth	Single Pair		All Pairs	
					DP?	Time	DP?	Time
SVT-Gauss	2	3	1.7	2.3	✓	1.6	✓	2.4
	5	6	3.3	2.7	✓	7.9	✓	76.7
	25	26	13.5	2.9	✓	441.3	–	O.M
SVT-Gauss-Leaky-1	5	6	3.3	1.7	×	1.0	×	1.4
	6	7	3.9	1.7	×	1.0	×	2.1
SVT-Gauss-Leaky-2	3	4	2.2	1.5	×	1.0	×	1.0
	6	7	3.9	1.7	×	1.0	×	1.1
SVT-Gauss-Ge	2	3	1.7	2.3	✓	1.3	✓	1.5
	5	6	3.3	2.7	✓	5.7	✓	72.3
	25	26	13.5	2.9	✓	501.6	–	O.M
SVT-Gauss-Ge-Leaky-1	5	6	3.3	1.7	×	1.0	×	1.2
	6	7	3.9	1.7	×	1.0	×	1.7
SVT-Gauss-Ge-Leaky-2	3	4	2.2	1.5	×	1.0	×	0.9
	6	7	3.9	1.7	×	1.0	×	1.0
SVT-Laplace	2	3	1.7	2.3	✓	5.1	✓	8.8
	5	6	3.3	2.7	✓	47.6	–	T.O
	11	12	6.4	2.8	✓	500.2	–	T.O
SVT-Laplace-Ge	2	3	1.7	2.3	✓	4.1	✓	8.8
	5	6	3.3	2.7	✓	46.0	–	T.O
	11	12	6.4	2.8	✓	497.8	–	T.O
SVT-Mix1	2	3	1.7	2.3	✓	2.4	✓	4.3
	5	6	3.3	2.7	✓	19.5	✓	285.4
	17	18	9.4	2.9	✓	365.7	–	O.M
SVT-Mix1-Ge	2	3	1.7	2.3	✓	2.3	✓	4.1
	5	6	3.3	2.7	✓	16.1	✓	261.0
	17	18	9.4	2.9	✓	343.6	–	O.M
SVT-Mix2	2	3	1.7	2.3	✓	7.2	✓	14.4
	5	6	3.3	2.7	✓	72.2	–	T.O
	10	11	5.9	2.8	✓	524.6	–	O.M
SVT-Mix2-Ge	2	3	1.7	2.3	✓	7.1	✓	13.4
	5	6	3.3	2.7	✓	67.7	–	T.O
	10	11	5.9	2.8	✓	506.5	–	O.M
Noisy-Max-Gauss	2	2	1.0	2.0	✓	1.0	✓	1.3
	3	4	2.0	2.5	✓	1.6	✓	3.1
	4	8	3.0	3.0	✓	37.8	✓	303.0
Noisy-Min-Gauss	2	2	1.0	2.0	✓	1.0	✓	1.3
	3	4	2.0	2.5	✓	1.6	✓	3.1
	4	8	3.0	3.0	✓	36.8	✓	303.6
Noisy-Max-Laplace	3	4	2.0	2.5	✓	13.1	✓	47.2
	4	8	3.0	3.0	–	T.O	–	T.O
Noisy-Min-Laplace	3	4	2.0	2.5	✓	9.8	✓	45.6
	4	8	3.0	3.0	–	T.O	–	T.O
m-Range-Gauss	1	7	3.0	2.5	✓	1.5	✓	1.8
	2	13	4.2	3.2	✓	171.2	✓	344.4
	3	19	5.2	3.8	–	T.O	–	T.O
k-Min-Max-Gauss	3	16	4.0	3.0	✓	2.1	✓	5.7
	4	28	5.1	3.4	✓	41.2	✓	335.7

Table 7: **Summary of Experimental Results for DiPApprox. The columns in the table are defined as follows: N is the input size of the program. DP? indicates whether the program is differentially private. Final States denotes the number of final states. $\overline{\lvert G \rvert}$ and Avg. Depth, respectively, denote the average number of conditions and the average nesting depth of integral expressions, per final state. Time is the average time (in seconds) to verify differential privacy, measured over three runs.** T.O **indicates a timeout (exceeding 10 minutes), and** O.M **denotes a run out of memory. Differential privacy checks were performed with $\epsilon = 0.5$ and $\delta = 0.01$, except for SVT-Gauss-Leaky-1, which uses $\epsilon = 8$. We**

Algorithm 27: k-Min-Max-Gauss

Input: $q[1:N]$
Output: $out[1:N]$

$\text{min}, \text{max} \leftarrow \mathcal{N}(q[1], \frac{4k}{\epsilon})$)
for $i \leftarrow 2$ **to** k **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4k}{\epsilon})$
  if $(\mathbf{r} > \text{max}) \wedge (\mathbf{r} > \text{min})$ **then**
    $\text{max} \leftarrow \mathbf{r}$
  else if $(\mathbf{r} < \text{min}) \wedge (\mathbf{r} < \text{max})$ **then**
    $\text{min} \leftarrow \mathbf{r}$
  end
  $out[i] \leftarrow \text{read}$
end
for $i \leftarrow k+1$ **to** N **do**
  $\mathbf{r} \leftarrow \mathcal{N}(q[i], \frac{4}{\epsilon})$
  if $(\mathbf{r} \geq \text{min}) \wedge (\mathbf{r} < \text{max})$ **then**
    $out[i] \leftarrow \perp$
  else if $(\mathbf{r} \geq \text{min}) \wedge (\mathbf{r} \geq \text{max})$ **then**
    $out[i] \leftarrow \top$
    exit
  else if $(\mathbf{r} < \text{min}) \wedge (\mathbf{r} < \text{max})$ **then**
    $out[i] \leftarrow \perp$
    exit
  end
end