A Programming Framework for Estimating Privacy Loss in Differential Privacy

Takumi Hiraoka
The University of Tokyo
Tokyo, Japan
hiraoka@eidos.ic.i.u-tokyo.ac.jp

Shumpei Shiina
Individual Researcher
Tokyo, Japan
s417.lama@gmail.com

Kenjiro Taura
The University of Tokyo
Tokyo, Japan
tau@eidos.ic.i.u-tokyo.ac.jp

1 Introduction

Differential privacy [6] is a mathematical concept for evaluating privacy protection. One of the most fundamental definitions within differential privacy is ϵ-differential privacy (ϵ-DP), where the parameter ϵ is referred to as "privacy loss."

Accurately estimating the privacy loss ϵ in differential privacy requires complex probabilistic computations, making estimation errors likely. These errors can occur not only when programmers implement algorithms but also when privacy experts design algorithms [9]. For example, in the case of the Sparse Vector Technique (SVT), it has been reported in later studies that the estimated privacy loss in earlier research was incorrect [4, 10].

Many methods for estimating privacy loss have been extensively studied [1–3, 5, 11, 12, 14–17]. However, depending on the algorithm, these methods may suffer from long computation times or low accuracy, and a practical estimation method has yet to be fully established.

In this study, we propose DPEST, a programming framework for efficiently estimating privacy loss in differential privacy. Programmers can obtain the privacy loss simply by writing the algorithm using DPEST. Our approach estimates privacy loss as precisely as possible by deriving the probability distribution of outputs. To compute output distributions, we employ either the sampling method or the probability accumulation method. Furthermore, a more optimal method may be adopted by considering the dependencies between random variables. This proposed estimation method enables more accurate privacy loss estimation with shorter execution times than existing methods for certain algorithms. Moreover, since DPEST supports low-level operations, it allows for flexible programming. We have confirmed that existing algorithms, used as evaluation benchmarks in previous research, can be implemented using DPEST.

The contributions of this paper are as follows:

(1) We propose DPEST, a new programming framework that efficiently estimates the privacy loss ϵ by computing the probability distribution of outputs.
(2) We design DPEST to support low-level programming, allowing flexible implementation of differential privacy-compliant algorithms.
(3) Through experimental comparisons, my method achieved shorter execution times and higher estimation accuracy than existing methods for certain algorithms.

This paper is organized as follows. In Section 2, we review the related work. In Section 3, we describe our proposed method in detail. Finally, in Section 4, we present experimental results comparing our approach with existing research.

2 RELATED WORK

In black-box methods [3, 5, 17], privacy loss is estimated based on hypothesis testing [5] or a classifier trained using machine learning [3, 17] on the obtained samples. A major limitation of these black-box methods is that they do not analyze the internal structure of the program, making optimization difficult.

In rigorous verification- and proof-based privacy loss estimation methods [1, 11, 12, 14–16], privacy loss is estimated by utilizing proofs or mathematical optimization solvers based on the program's internal structure. These methods rely on high-performance symbolic solvers or theorem provers which can lead to excessive computation times or inapplicability to certain algorithms. Furthermore, there is an additional issue in that the boundary between applicable and non-applicable programs is not clearly defined.

3 METHOD

To address the issues described in Section 2, this paper proposes DPEST, a domain-specific language (DSL) designed for efficient privacy loss estimation. DPEST has the following features:

(1) It is a programming framework designed for efficiently estimating privacy loss.
(2) Unlike existing verification- and proof-based estimation methods that rely on mathematical optimization solvers or theorem provers, DPEST does

not depend on such solvers, ensuring that estimation is always possible within the scope of programs expressible in DPEST.
(3) As a DSL, it supports low-level operations, allowing for flexible programming.
(4) By leveraging algorithm-specific characteristics for optimization, DPEST enables more efficient privacy loss estimation.

Hereafter, Section 3.1 introduces DPEST as a programming framework for estimating privacy loss in programs, and the way to use and Section 3.2 describes the privacy loss estimation mechanism of DPEST.

3.1 DPEST: Programming Framework

DPEST is implemented as a Python library, allowing algorithms to be written using both Python's built-in operations and DPEST's specialized operations for transforming random variables. Variables with noise added to satisfy differential privacy can be treated as random variables. DPEST provides basic random variable transformation operations, such as like *Add*, *Mul*, *Br*, *Case*, and *ToArray*. By combining these operations as transformations of random variables, we were able to describe all the algorithms used as experimental subjects in existing studies [3].

As an example, when the NoisyMaxLap[5, Alg.7] algorithm— which adds Laplace noise to each element of an array and returns the maximum value— is implemented using DPEST, it results in the source code shown in Program 1. The algorithm itself is implemented in the

Program 1: Implementation of the NoisyMaxLap Algorithm Using DPEST

```
# Omitted: Importing necessary modules
def noisy_max_lap():
    INPUT_ARR_SIZE, eps, sens = 5, 0.1, 1
    LapArr = list(laplace_extract(InputArray(
         INPUT_ARR_SIZE, adj="inf"), sens/eps))

    Y = LapArr[0]
    for i in range(1, INPUT_ARR_SIZE):
        Y = Br(Y, LapArr[i], Y, LapArr[i])
    return Y

if __name__ == "__main__":
    ConfigManager.load_config()
    alg = noisy_max_lap()
    privacy_loss = eps_est(alg)
```

noisy_max_lap function, and by applying the `eps_est` function to its return value, the algorithm is analyzed, and the privacy loss is computed. Additionally, ConfigManager.load_config() is a function that loads various parameters, such as the number of samples and the number of bins in the histogram.

Algorithm 1 Algorithm for Estimating Privacy Loss

```
function EPS_EST(alg)
    max_eps ← 0
    # Step1. Generate a calculation graph with injected dependency information from the algorithm
    g ← gen_calc_graph(alg)
    for each (D, D′) in input_list do
        # Step2. Insert the input and calculate the probability distribution
        Y, Y′ ← calc_pdf(g, D, D′)
        # Step3. Search for the point where the probability ratio is maximized
        eps ← search_max_eps(Y, Y′)
        if max_eps < eps then
            max_eps ← eps
        end if
    end for
    return max_eps
end function
```

3.2 Privacy Loss Estimation Mechanism of DPEST

We'll explain the mechanism of the `eps_est` function for estimating the privacy loss of an algorithm.

The fundamental approach of DPEST for privacy loss estimation is to treat values as random variables once noise is added. This allows the final output to be handled as a random variable as well. The method then derives or approximates the probability density function (PDF) or probability mass function (PMF) for two adjacent input datasets and computes the ratio of these densities or probabilities, searching for the point where the ratio is maximized.

More specifically, the `eps_est` function estimates privacy loss by taking an algorithm `alg` as an argument and executing the following three-step process, as described in Algorithm 1. In Step 1, the function takes an algorithm represented by the variable `alg` as input and generates a computational graph that incorporates the dependencies among the arguments of each operation. If the arguments of an operation are independent, the probability distribution of the operation's output is computed using the probability accumulation method, as described in Section 3.2.2. If the arguments are not independent, the sampling method described in Section 3.2.1 is used instead.

In Step 2, the computational graph g of the algorithm is executed with two adjacent input datasets $\mathcal{D}$ and $\mathcal{D}'$, yielding the corresponding output probability distributions Y and Y'. According to the definition of ϵ-differential privacy (ϵ-DP), ideally, the algorithm should be executed for all possible input datasets to compute

their output probability distributions. However, since it is computationally expensive, we adopted adjacent input dataset patterns as proposed in previous studies [3, 5, 17], which were empirically chosen.

In Step 3, the function finds the point where the ratio of the two output distributions Y and Y' is largest.

In the following sections, we explain the methods for obtaining the output probability distribution.

3.2.1 Sampling Method This method repeatedly performs sampling from the original probability distribution, executes the program using these samples to obtain outputs, and constructs a histogram based on the outputs to approximate the probability mass function (PMF). The number of samples, `SAMPLING_NUM`, and the number of histogram bins, `B`, are given as parameters. If `SAMPLING_NUM` is sufficiently large, increasing `B` allows for a more accurate estimation of the PDF. However, if `SAMPLING_NUM` is small, a larger `B` results in fewer samples per bin, leading to higher variance and lower estimation accuracy.

3.2.2 Probability Accumulation method In DPEST, when the random variables used as arguments in an operation are independent, the probability accumulation method is used to obtain the output probability distribution by accumulating the probabilities of mapped values. To support this method, DPEST operations enable transformations of random variables. If a program variable X represents a random variable, its PDF or PMF is represented in a dictionary format like $f_X(x) = \{x_1 : p_1, x_2 : p_2, \ldots, x_n : p_n\}$.

For example, when considering the sum of two independent random variables, the resulting probability distribution is obtained through convolution. Convolution can be interpreted as the operation of summing the original probabilities that map to a given value. This operation itself can be seen as an accumulation of mapped probabilities. In DPEST, the sum of two random variables can be computed using the operation, $Add(X_1 : Pmf,\ X_2 : Pmf)$. The resulting probability distribution is given by the formula, $f_Y(y) = \sum_x f_{X_1}(x) f_{X_2}(y - x)$. To ensure consistent value intervals for the random variables, DPEST employs spline interpolation before performing the computation. The final output probability distribution for the sum is then obtained using the following procedure.

$$f_y = \Big\{ x_{1i} + x_{2j} : \sum_x f_{X_1}(x)\, f_{X_2}\big((x_{1i} + x_{2j}) - x\big) \;\Big|\; 1 \le i \le n,\ 1 \le j \le m,\ i, j \in \mathbb{Z} \Big\}.$$

In this method, errors in estimation basically may arise due to two factors: approximating the original probability density function (PDF) as a probability mass function (PMF) using histograms and limiting the range of random variables. Especially, for algorithms like OneTimeRAPPOR and RAPPOR [7], where the random variables representing the noise distribution take only categorical values and directly handle probability mass functions from the beginning, no estimation error occurs. Given d as the number of dimensions in the input data and m as the number of possible values for each random variable, the estimation can be performed with a computational complexity of only $O(m^d)$, which is lower than that of the sampling method $O(\texttt{SAMPLING\_NUM})$ when d and m is small.

3.2.3 Optimization It is possible to optimize specific algorithms by leveraging dependencies between random variables and the characteristics of the algorithm. For example, in the case of an algorithm that returns an array of random variables, if all elements in the array are independent random variables, the maximum ratio of each random variable's probability distribution can be computed individually, and their product can be taken to obtain the maximum ratio of the original output probability distribution.

4 EVALUATION

We compared the accuracy and execution time of DPEST with existing privacy loss estimation methods, StatDP [5] and DP-Sniper [3]. In the experiments, we used the same algorithms as those employed in the DP-Sniper [3] paper. The hyperparameters of DPEST were configured so that its execution time was on the same order of magnitude as that of existing methods. The experimental results are shown in Figure 1 and Figure 2.

For the algorithms LaplaceMechanism, NoisyHist1 [5, Alg.9], NoisyHist2 [5, Alg.10], LaplaceParallel, NoisyMaxLap [5, Alg.7], NoisyMaxExp [5, Alg.8], OneTimeRAPPOR, and RAPPOR, where the probability accumulation method was used by leveraging the independence between random variables, we achieved a significant reduction in execution time while maintaining high accuracy. Additionally, for algorithms such as NoisyArgMaxLap [5, Alg.5], NoisyArgMaxExp [5, Alg.6], SVT1/2/4/5/6 [10], and TruncatedGeometric [8, Ex. 2.2], where the output is categorical and the sampling method was used, we achieved comparable accuracy and execution time to existing methods. On the other hand, for algorithms such as SVT3 [10], SVT34Parallel ($a \mapsto (\text{SVT3}(a), \text{SVT4}(a))$), NumericalSVT [13, Fig.10], and PrefixSum [13, App. C.3], which include floating-point values in their output, the estimated privacy loss was observed to be higher. This is due to the insufficient number of samples relative to the number of bins in the estimation process. Improving it remains a future challenge for DPEST.

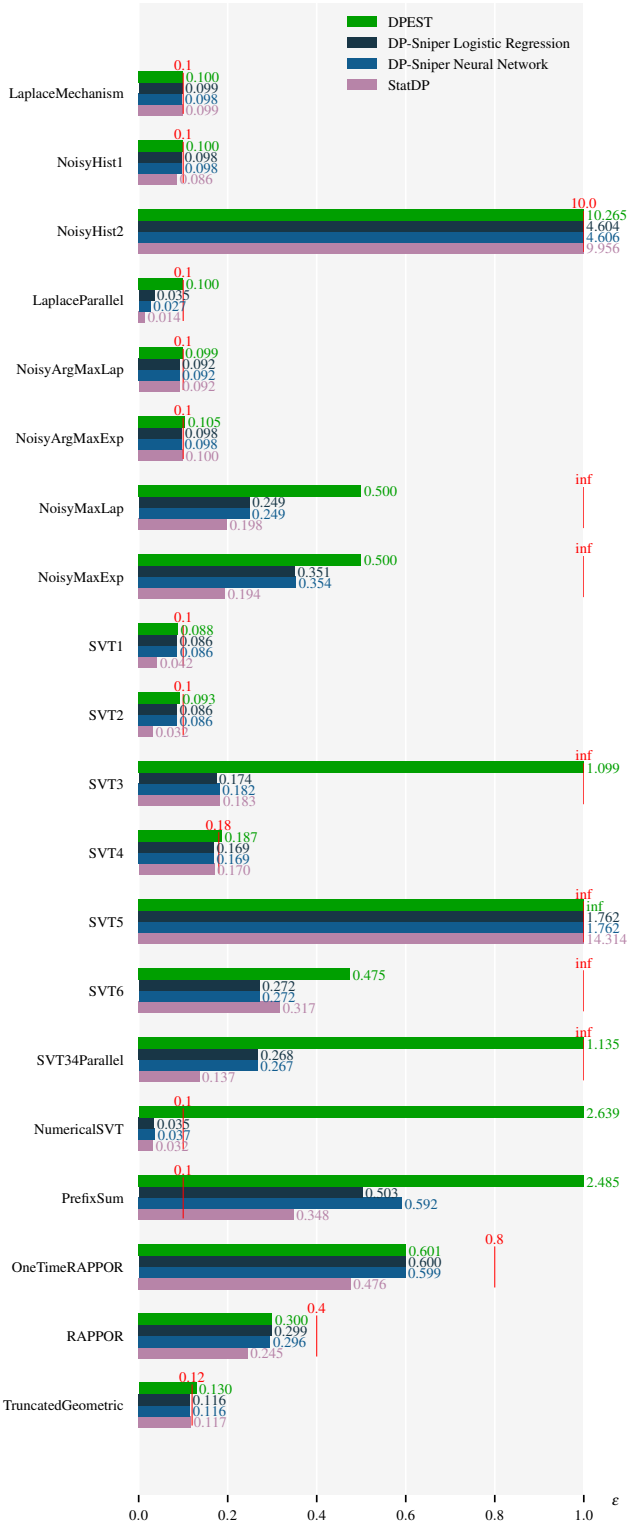

Figure 1: Comparison of the Accuracy of DPEST, DP-Sniper, and StatDP

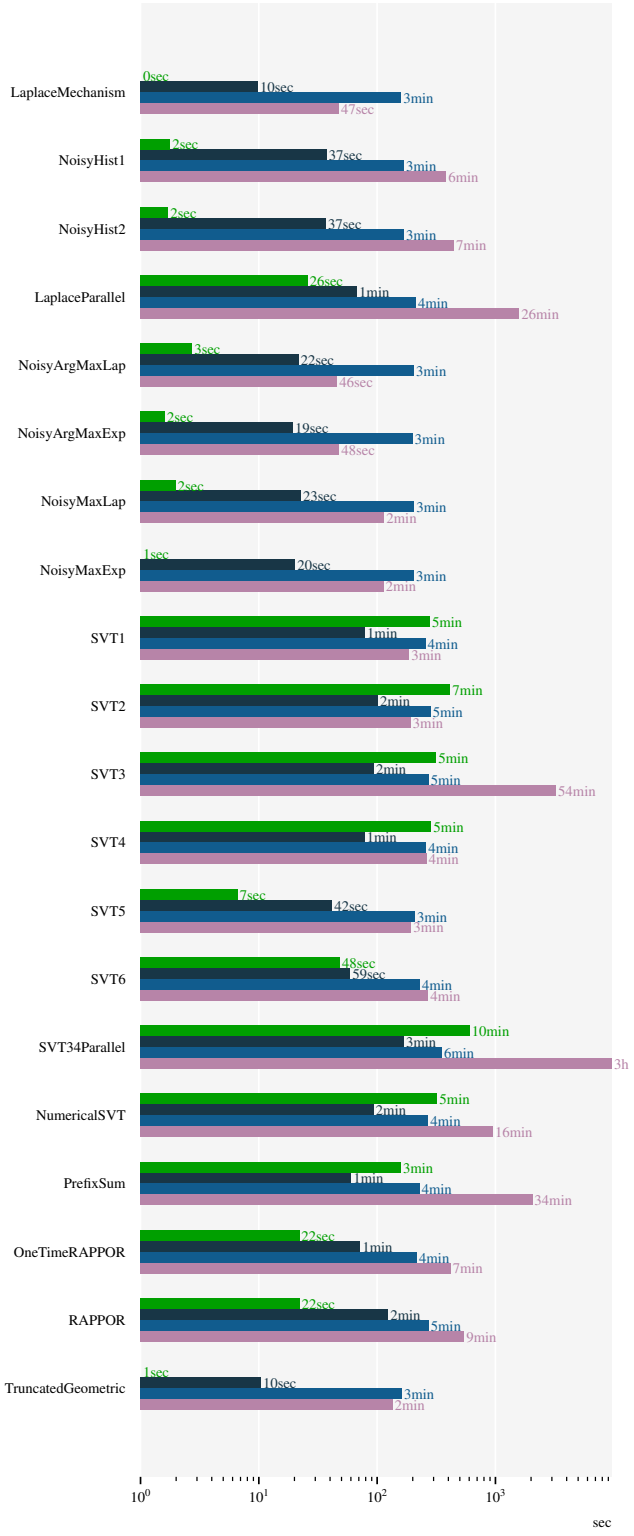

Figure 2: Comparison of the Execution Time of DPEST, DP-Sniper, and StatDP

References

[1] Aws Albarghouthi and Justin Hsu. 2017. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.* 2, POPL, Article 58 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158146

[2] Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin Vechev. 2018. DP-Finder: Finding Differential Privacy Violations by Sampling and Optimization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 508–524. https://doi.org/10.1145/3243734.3243863

[3] Benjamin Bichsel, Samuel Steffen, Ilija Bogunovic, and Martin Vechev. 2021. DP-Sniper: Black-Box Discovery of Differential Privacy Violations using Classifiers. In *2021 IEEE Symposium on Security and Privacy (SP)*. 391–409. https://doi.org/10.1109/SP40001.2021.00081

[4] Yan Chen and Ashwin Machanavajjhala. 2015. On the Privacy Properties of Variants on the Sparse Vector Technique. arXiv:1508.07306 [cs.DB] https://arxiv.org/abs/1508.07306

[5] Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. 2018. Detecting Violations of Differential Privacy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM. https://doi.org/10.1145/3243734.3243818

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography* (New York, NY) *(TCC'06)*. Springer-Verlag, Berlin, Heidelberg, 265–284. https://doi.org/10.1007/11681878_14

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

[8] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. 2009. Universally Utility-Maximizing Privacy Mechanisms. arXiv:0811.2841 [cs.DB] https://arxiv.org/abs/0811.2841

[9] Xiyang Liu and Sewoong Oh. 2019. Minimax Rates of Estimating Approximate Differential Privacy. arXiv:1905.10335 [cs.IT] https://arxiv.org/abs/1905.10335

[10] Min Lyu, Dong Su, and Ninghui Li. 2017. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.* 10, 6 (Feb. 2017), 637–648. https://doi.org/10.14778/3055330.3055331

[11] 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 *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 919–938. https://doi.org/10.1145/3372297.3417282

[12] 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 '19)*. ACM, 655–669. https://doi.org/10.1145/3314221.3314619

[13] 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 '19)*. ACM, 655–669. https://doi.org/10.1145/3314221.3314619

[14] Danfeng Zhang and Daniel Kifer. 2016. AutoPriv: Automating Differential Privacy Proofs. (07 2016). https://doi.org/10.48550/arXiv.1607.08228

[15] Danfeng Zhang and Daniel Kifer. 2017. LightDP: towards automating differential privacy proofs. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL '17)*. ACM, 888–901. https://doi.org/10.1145/3009837.3009884

[16] Hengchu Zhang, Edo Roth, Andreas Haeberlen, Benjamin C. Pierce, and Aaron Roth. 2020. Testing differential privacy with dual interpreters. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 165 (Nov. 2020), 26 pages. https://doi.org/10.1145/3428233

[17] Yushan Zhang, Xiaoyan Liang, Ruizhong Du, and Junfeng Tian. 2024. DP-Discriminator: A Differential Privacy Evaluation Tool Based on GAN. In *Proceedings of the 21st ACM International Conference on Computing Frontiers* (Ischia, Italy) *(CF '24)*. Association for Computing Machinery, New York, NY, USA, 285–293. https://doi.org/10.1145/3649153.3649211